T/S 1500 T/S 1000 ZX-81

Explorer's Guide to the Timex Sinclair 1500/1000

Mike Lord

Discover the limitless capabilities of your computer!

BookWare T.M.

For the Timex Sinclair 1500, Timex Sinclair 1000 and ZX-81 Computers

ENTERED AUG - 2 1989

The Explorer's Guide to the Timex® Sinclair 1500/1000™

Written by Mike Lord
Including material by John Durst,
S. J. Newett, and Lee Gentry

A Reston Computer Group Book
Reston Publishing Company, Inc.
A Prentice-Hall Company
Reston, Virginia

This book is dedicated
to Clive Sinclair who
made it all possible.

ISBN 0-8359-1816-5

Copyright 1982 by Timedata Ltd.

This edition of THE EXPLORER'S GUIDE TO THE TIMEX SINCLAIR 1500
is produced by Reston Publishing Company, Inc., A Prentice-Hall
Company, 11480 Sunset Hills Road, Reston, Virginia 22090.
Sales of this edition are restricted to North America.

This book, and the programs in it, can be used with the Timex
Sinclair 1500 and 1000 computers, and with the Sinclair ZX81.
References in the book to the ZX81 apply equally to the
T/S 1500 and T/S 1000.

Originally published by Timedata Ltd.,
57 Swallowdale, Basildon, Essex, U.K.
Originally published as THE EXPLORER'S GUIDE TO THE ZX81
AND TIMEX SINCLAIR 1000.

All rights reserved. No part of this book may be
reproduced by any means including electronic,
mechanical, photocopying, recording or otherwise
without the prior permission of the publishers.

The ZX81 design and the contents of the ZX81
ROM are the copyright of Sinclair Research Ltd.
Timex® Sinclair 1000™ is a registered
trademark of the Timex Corporation.

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

CONTENTS

3 Introduction

5 Chapter 1 ; Programming Aids

6 Saving Space
11 Fixed Headings
11 RAMtop Check
12 Spacey Strings
12 DIM Strings
13 Clear Before SAVE
13 VAL Functions
13 Pretty Printing
15 POKE 16418
16 Permanent Notice
16 Other BASICs

26 Chapter 2 ; Machine Language

27 Binary and Hexadecimal
31 Using USR
32 Machine Language
33 Where To Put It

39 Chapter 3 ; Discovering the ROM

40 ROM Routines
42 ROM Tables
43 LOAD & SAVE
43 The SAVE Routine
45 The LOAD Routine
47 Display
58 Keyboard Scanning

61 Chapter 4 ; Some Games, and Other Novelties

62 Weekday
63 Dynamite
65 Sums Tester
66 ZX Sofft Shoppes
69 Copycat
70 Bindechex
71 Roman-Arabic
72 Arabic-Roman
73 Moonlander
74 Things
77 Buzz
78 Bob-Sleigh
79 Recipes
80 SAM

81 Cambridge Crypto
81 CR?S??ORD
82 Starburst
83 All Change
84 Birds
85 Alien Attack
87 Carrier
88 Calendar
89 Swarm
90 Bulbs
91 Dragon
94 Buzzbox
95 Mixup
97 NIM
99 Caterpillar

102 Chapter 5 ; Applications

103 Application Notes
105 Std. Dev.
106 Ladder Analysis
111 G.P.G.P.
115 Bank
119 Linear Programming
123 Simultaneous Equations
125 Members

130 Chapter 6 ; Utilities

131 Decimal Peeker
132 Variable Peeker
133 RAMtest
133 ROMtest
134 Renumber
135 Catalogue

137 Chapter 7 ; Hardware

138 Battery Power
138 Cool It
139 Getta Betta Jack
139 Support Your RAM
139 Spiky Mains
140 Another Keyboard
140 Push To Reset
141 Connecting a Monitor
142 From the Speaker
142 Recording
144 Memory Map
146 Pseudo-ROM
149 1-4K RAM Extension
150 24 Line I/O
153 16K Byte RAM

Introduction

Welcome, fellow explorer, to the world of the ZX81.

And what a fascinating world it is. One in which you have total control. A world without ambiguities, without rebellion, without questions, where your word is law and you commands are executed exactly and to the letter - even though what you actually told it to do may not have been quite what you meant !

It is interesting to reflect that in the mid 1960's, the cheapest computer available cost more than three year's wages for the average person, was about 2' cube, and needed a slow and noisy teleprinter to communicate with the user. Even by the end of the 1970's, when the microcomputer revolution was well under way, you would still have had to lay out 2 or 3 month's salary to get a computer that would recognise even a primitive form of BASIC. But now, thanks mainly to the vision and drive of Clive Sinclair, the average weekly wage packet will buy you a machine with enough power and flexibility to perform a proper job of work as well as providing an almost endless source of interest for the owner.

Now, for a few words about the contents of this book. It is not intended to be a beginner's guide to BASIC, as many excellent books already exist on this subject. Neither is it an introduction to the ZX81; the ZX81 BASIC Manual does this job well.

Instead, this guide attempts to carry the reader on from where the ZX81 BASIC Manual leaves off, by giving a variety of programs to run, and - more importantly - a wealth of detailed information about the computer and how to get the best out of it.

In Chapter 1, we look at BASIC programming techniques not given in the ZX81 Manual, and at converting programs written in other versions of BASIC to run on the ZX81. Chapter 2 then gives an introduction to the complex and detailed world of Machine Language, which leads naturally on to an examination of the main ROM routines in chapter 3.

All of this knowledge is then put to use in the next three chapters which contain examples of games programs, business and engineering applications, and some useful utility routines.

Finally, chapter 7 looks at ways of improving your ZX81 hardware.

Whether your interests lie in games programs, in the business or engineering applications of the ZX81, or even in attacking it with a soldering iron, you should find something of interest in these pages.

But I hope that the most valuable thing the reader will find in this book is the impetus to carry on his own investigations, to

write his own programs, and to pursue his own explorations into the mysteries of this marvelous machine.

Some notes about the program listings in this book;

- Programs are intended to be run in the SLOW mode unless otherwise stated.
- Programs which will fit into 1K RAM are so marked. The other programs were written to use 16K but in practice most will fit into a ZX81 with only 4K of RAM.
- The number zero is written as 'Ø' throughout the listings to distinguish it from the letter O.
- Generally, only single letter variable names have been used in the programs; where names longer than one character have been used then this is mentioned in the notes accompanying the program listing. Usually, therefore, any time you see two or more letters together then they represent a *keyword*, to be entered by a single keystroke.
- To avoid confusion, the 'double character' keyword "" has not been used.
- The '<=','>=' and '<>' keywords have, however, been used; remember that they are entered by a single keystroke.
- Graphics characters have been drawn as they appear, thus '▜' represents the graphics character on key Q. Where there may be some difficulty in recognising the graphics shapes, then the notes below the program listing will tell you which keys should be used.
- The inverse (white on black) characters have been avoided wherever possible to simplify the listings - where they have been used then an explanation is given in the program notes. But please feel free - once you have the program running - to change it to incorporate inverse characters wherever you feel that they will add interest to the display.

The later ZX81's have the legend 'ENTER' instead of 'NEWLINE' on the rightmost key, and 'DELETE' rather than 'RUBOUT' on the top right key. Please make any mental corrections necessary as you read through the book if you have one of these models, and please forgive us for not showing both versions throughout the book!

Mike Lord

Chapter 1

Programming Aids

THE PROGRAMMER'S SONG OF TRIUMPH

The first version I wrote was the first that I sold.
The second one worked. But then I was bold
And re-wrote it again. And again. And again.
Each issue was better by far than the last.
And now its so short, and so neat, and so fast,
You can't buy it for dollars, but only for Yen.

Saving Space

The major limitation of the basic ZX81 is the amount of RAM available. Of the 1K (1024) bytes, 125 bytes are used for the ZX81's System Variables, between 25 and 793 bytes are used for the Display File, and a variable amount is taken up by the Calculator, Machine and Gosub stacks. You can end up with very little space left for your program and its variables.

Display Small

As the Display File can be very greedy for RAM, we should look first at how its appetite can be curbed. If you have 4K bytes or more, then there is nothing you can do; as the Display File is then always kept in its fully expanded 793 byte form.If, however, you have less than 4K bytes of RAM, then read on ;

The judicious use of CLS statements can help by collapsing the Display File to the minimum 25 bytes. Having done this, then concentrate your efforts on keeping the Display File as small as possible. PRINT and PLOT as little as possible, and when you do then do it on the left hand side of the screen rather than in the middle or on the right. This is because whenever anything is displayed on the right hand side of the screen, the Display File has to hold all of the spaces leading up to it, whereas the ZX81's clever design doesn't need the Display File to hold trailing spaces in a line. Note also that if you delete anything from the display by over-printing it with spaces, then the spaces remain in the Display File - taking up valuable RAM - even though the resulting line may be totally blank.

Think Small

Having cut the Display File down to size, it is time to look at the program itself. Obviously the algorithm chosen will have a major effect on the length of the program, as will the number of 'frills' you include - such as checking inputs for validity, formatting outputs, and providing user prompts - and it is impossible to give any guidelines for these factors except perhaps to say that re-thinking your ideas will usually prove fruitful. Remember the Old Programmer's saying;

"He who thinks most bytes least"

It is usually possible to save several bytes by taking a detailed look at each line of the program and - with the aid of the table at the end of this section - seeing if alternative renderings would occupy less space. Some things to look for are given in the following paragraphs.

Small Tricks

Numeric constants - such as 'Ø' or '99.99' use one byte for each character *plus six bytes to hold the binary equivalent of the number*. Thus the line;

```
10 IF A>55 THEN LET A=55
```

uses 28 bytes, whereas

```
10 LET Z=55
11 IF A>Z THEN LET A=Z
```

takes a total of 30 bytes for both lines, plus 6 bytes in the Variable Area to hold the new variable Z. This appears to be a backward step, but if the constant 55 were used in other places in the program, and if the variable Z could perhaps be re-used, then an overall saving could be made.

Alternatively, we can often use the function CODE to generate a numeric constant - although at the cost of making the program more difficult to understand. Noting that CODE "R" equals 55 and occupies only 4 bytes of program space, the example given above could be written as;

```
10 IF A>CODE "R" THEN LET A=CODE "R"
```

and this version would take only 20 bytes, a saving of 8 bytes over the original.

The VAL function is also useful, as for example

```
10 LET A=VAL "356"
```

uses 3 less bytes (but takes longer to execute) than;

```
10 LET A=356
```

If you want the numbers 'Ø' or '1', then short equivalents are NOT PI and PI/PI, saving 5 & 4 bytes respectively.

Variables should be re-used wherever possible, especially FOR loop control variables, to hold down the size of the Variable Area of RAM.

Multi-character variable names may give a clue to how the program works, but they use up RAM. Use only single character names wherever possible. Similarly, explanatory REM statements are better hand-written on your copy of the program listing.

One long PRINT statement can often take less memory than two short ones.

Make the program finish at the last line, to remove the need for a STOP statement.

AND is often a slightly shorter alternative to IF - - THEN. For example;

```
10 IF A>B THEN PRINT "BIG"
```

takes 16 bytes, whereas;

```
10 PRINT "BIG" AND A>B
```

takes 15.

Using parentheses in arithmetic and string expressions uses RAM not only for the '(' and ')' symbols but also as a temporary store for the intermediate result when the program is run. Similarly, GOSUBs use extra RAM when the program is run - to hold the return addresses.

If an expression is used several times within a program, it may be possible to save space by defining a string as that expression and then using VAL to execute it as required.

If numeric or string variables have to be set to fixed values at the start of your program, then a potent way of saving space is by assigning their values by direct commands - even to the extent of DIMensioning any arrays directly. But remember that you can't RUN such a program, but must start it with GOTO to preserve the assigned values. To see how this works, enter the program;

```
10 SAVE "TEST"
20 PRINT A,B,C(1),C(2)
30 PRINT "AGAIN ?"
40 INPUT D$
50 IF D$="Y" THEN GOTO 20
```

Then enter the following lines as direct commands (i.e. without line numbers);

```
CLEAR
DIM C(2)
LET A=1
LET B=2
LET C(1)=3
LET C(2)=4
```

Then GOTO 20 should display the stored values, and GOTO 10 will save the program and the values on tape so that if it is subsequently LOADed, the values will be recovered and then printed out. This technique should - however - be used with caution as it becomes very difficult to debug a program containing 'invisible' data.

Taking slices of a fixed string is often an economical way of dealing with a list of fixed data (characters, strings or numbers), and is less error-prone than using the 'invisible data' technique. For example;

```
10 DIM A(4)
20 DIM B$(4)
30 DIM C$(4,3)

50 FOR I=1 TO 4
60 LET A(I)=VAL "1248"(I)
70 LET B$(I)="ZX81"(I)
80 LET C$(I)="ROMRAMZ80IBM"(3*I-2 TO 3*I)
90 NEXT I
```

will load the elements of the array A() with the numbers 1,2,4,8 the elements of the character array B$() with the characters Z,X,8 and 1 , and the string array C$() with the words ROM RAM Z80 and IBM. See the programs "Roman - Arabic" and "Arabic - Roman" for other examples.

Shrunken Arrays

It takes 5 bytes to hold each value in a numeric array, but if you only want to store a limited range of values, then you can save space by storing the values in a character array, using the CODE and CHR$ functions. For example, to input and store 20 values each in the range 0 to 255;

```
10 DIM A$(20)
20 FOR B=1 TO 20
30 INPUT X
40 LET A$(B)=CHR$ X
50 NEXT B
```

and to print them out;

```
100 FOR B=1 TO 20
110 PRINT CODE A$(B)
120 NEXT B
```

A larger range of values can be accommodated by taking two or three bytes of a character array to hold each value.

How Much

If you want to find out how much RAM has been used, remember that;

```
PRINT PEEK 16396 + 256*PEEK 16397
```

gives the address of the beginning of the Display File, which follows immediately after the end of the Program Area.

```
PRINT PEEK 16400 + 256*PEEK 16401
```

gives the address of the start of the Variable Area, which follows immediately after the end of the Display File.

```
PRINT PEEK 16404 + 256*PEEK 16405
```

gives the address of the first free byte after the end of the variables area.

But also remember that the RAM above the end of the Variables area cannot be considered completely free, as the ZX81 uses this area for various stacks and you may (by re-assigning RAMTOP) have reserved some for USR routines.

If you want to investigate the way that programs,data and the display file are held in RAM, the "Peeker" programs given elsewhere in this book will be useful.

RAM Space

RAM SPACE USED BY VARIOUS ZX81 BASIC ELEMENTS

The number of bytes needed to store a program line is not the same as the number of characters shown when the line is LISTed, but may be calculated from the figures given below,

	Bytes used in program area
Any number except a line number at the start of a line.	6 plus one for each character in the LISTed number.
Any keyword, function, statement or operator.	1
Any other graphics or alpha-numeric character appearing in the LISTed form of the line.	1

In addition, each program line takes 2 bytes to store the line number (in binary form), 2 bytes to store the length of the stored line, and one byte for the NEWLINE code at the end of the line.

As an example, the line;

```
                10 LET A(I) = 8
takes;             1   4   1 7      plus 5 = 18 bytes
```

Each variable used also takes up space in the Variables Area of RAM;

	Bytes taken in variable area
FOR control variable.	18
Other numeric variable.	5 plus one byte for each character in the variable's name.
Numeric array.	4 plus 2 bytes per dimension plus 5 bytes per element.
String variable.	3 plus 1 byte per character in the string.
Character array.	4 plus 2 bytes per dimension plus 1 byte per element.

Other Hints

Fixed Headings

For reasons not revealed to the author, the CLS command counts 24 NEWLINEs backwards from the end of the display to find the beginning. This means that if we POKE an extra NEWLINE code into the Display File, then CLS will not affect the first line. As a demonstration, run;

```
 10 PRINT "TITLE LINE*"
 20 POKE (11+PEEK 16396+256*PEEK 16397),118
 30 FOR A=1 TO 3
 40 FOR B=1 TO 21
 50 PRINT AT B,0;B
 60 NEXT B
 70 PAUSE 50
 80 CLS
 90 NEXT A
100 POKE (11+PEEK 16396+256*PEEK 16397),0
110 CLS
```

The NEWLINE code (118) is POKEd in place of the '*' in the title line by line 20, and later removed by line 100.

Note that this program actually prints on the 23rd. display line.

RAMTOP Check

If you have a program which needs the System Variable RAMTOP to be changed before LOADing, it is perhaps worth including a check that this has been done at the start of the program,e.g;

```
10 IF PEEK 16388+256*PEEK 16389=17388 THEN GOTO 20
11 PRINT "RAMTOP WRONG"
12 STOP
20 - - - -
```

(The correct value for RAMTOP is assumed to be 17388)

Spacey Strings

A string consisting of a defined number of spaces is often useful, and the easiest way to generate it is by using a DIM statement. For example;

```
DIM A$(32)
```

creates a string A$ of 32 spaces.

DIM Strings

If you are using a string array, then remember that the length of each string in the array is always fixed (by the DIM statement). Thus;

```
10 DIM A$(2,5)
20 LET B$="ABC"
30 LET A$(1)=B$
40 PRINT A$(1),LEN A$(1),B$,LEN B$
```

will print out

```
ABC             5
ABC             3
```

showing that A$(1) always retains the length of 5 characters.

When a string element of a string array is assigned a new value, then the new value is either truncated or packed out at the end to bring it to the correct length.

This can be awkward when trying to search through a string array for - say - a matching name, as the comparison will have to take account of any 'spare' spaces. For example, the program;

```
10 DIM A$(2,5)
20 LET A$(1)="ME"
30 IF A$(1)="ME" THEN PRINT "FOUND ME"
```

won't actually print anything, as the two strings being compared in line 30 are of different lengths.

One solution to the problem lies in dimensioning a test string to have the same length as the strings in your array, and then using that test string in the comparison;

```
10 DIM A$(2,5)
20 DIM X$(5)
30 LET A$(1)="ME"
40 LET X$="ME"
50 IF A$(1)=X$ THEN PRINT "FOUND ME"
```

will work as expected.

CLEAR before SAVE

If you don't need to save the values of the variables with your program, then a CLEAR command before the SAVE reduces the amount of information the ZX81 has to put onto the tape. This cuts down the time taken to SAVE, and later to LOAD, the program, and so reduces the chance of an error.

VAL Functions

If an expression is repeated several times within a program, then it may be worth defining it as a string, then using VAL to execute it when required. For example, suppose the line

```
LET Z=SQR (X*X+Y*Y)
```

was used several times, then we could insert a line at the start of the program;

```
LET Z$="SQR (X*X+Y*Y)"
```

then subsequently write;

```
LET Z=VAL Z$
```

Pretty Printing

To print A$ centrally on a line, use;

```
PRINT TAB 16-LEN A$/2;A$
```
or
```
PRINT AT n,16-LEN A$/2;A$
```

To right-justify A$, use;

```
PRINT TAB 32-LEN A$;A$
```
or
```
PRINT AT n,32-LEN A$;A$
```

Printing a column of numbers so that they line up properly is easy if they are all integers, as illustrated by the following;

```
10 LET A=0
20 FOR B=1 TO 9
30 LET A$=STR$ A
40 PRINT TAB 12-LEN A$;A$
50 LET A=10*A+B
60 NEXT B
```

```
 0
 1
 12
 123
 1234
 12345
 123456
 1234567
 12345678
```

But if decimal fractions are involved, a special routine is needed to line up the decimal points, as lines 30-70 of the demonstration program below;

```
10 LET A=PI*.00001
20 FOR B=1 TO 18
30 LET A$=STR$ A
40 FOR C=1 TO LEN A$
50 IF A$(C)="." THEN GOTO 70
60 NEXT C
70 PRINT TAB 14-C;A$
80 LET A=A*10
90 NEXT B
```

```
      .00003141592
      .00031415927
      .00314159270
      .03141592700
      .31415927000
     3.14159270000
    31.41592700000
```

We often want to round results off to a maximum number of digits after the decimal point. The INT function can be used to do this as for example in the following routine which rounds to 3 decimal places after the point;

```
10 LET A=PI/10000
20 FOR B=1 TO 10
30 PRINT INT (.5+A*1000)/1000
40 LET A=A*10
50 NEXT B
```

```
0
.003
.031
0.314
3.142
31.416
314.159
3141.593
31415.927
314159.27
```

The decimal points could be lined up if required using the technique given earlier. However, this form of output is not ideal for monetary values such as pounds and pence, or dollars and cents; it would show for example one dollar and thirty cents as '1.3', which although mathematically correct, is not what the customer is used to.

To print in a more acceptable format, a routine such as that included in the demonstration program below is needed. This example also lines up the decimal points;

```
10 LET A=PI/1000
20 FOR B=1 TO 10
30 LET A$=STR$ INT (.5+A*100)
40 IF LEN A$=1 THEN LET A$=A$+"0"
50 IF LEN A$=2 THEN LET A$="0"+A$
```

```
60 PRINT TAB 10-LEN A$;A$( TO LEN A$-2);".";A$(LEN A$-1 TO )
70 LET A=A*10
80 NEXT B
```

```
     0.00
     0.30
     0.31
     3.14
    31.42
   314.16
  3141.59
 31415.93
314159.27
3141592.70
```

This routine is not quite complete, as it won't handle negative numbers properly. To allow it to do so, the following lines should be added;

```
35 LET B$=(" " AND A$(1)<>"-")+("-" AND A$(1)="-")
36 IF B$="-" THEN LET A$=A$(2 TO )
```

and line 60 changed to;

```
60 PRINT TAB 11-LEN A$;B$ A$( TO LEN A$-2);".";A$(LEN A$-1 TO )
```

or - if you want the minus sign to be printed *after* the number, in true accounting style;

```
60 PRINT TAB 10-LEN A$;A$( TO LEN A$-2);".";A$(LEN A$-1 TO );B$
```

POKE 16418

System variable DF-SZ (16418) holds the number of lines (normally 2) in the bottom part of the screen. POKEing the value 0 into this location lets you PRINT in these two lines, e.g;

```
10 POKE 16418,0
20 FOR A=0 TO 23
30 PRINT AT A,10;"LINE ";A
40 NEXT A
```

But you must POKE the value 2 (or greater) back again before any INPUT or SCROLL command, or the system will crash.

POKEing a value greater than 2 limits the number of lines which are moved up by a SCROLL; try the example given above with the following lines added;

```
50 POKE 16418,10
60 FOR A=1 TO 4
70 SCROLL
80 NEXT A
```

Permanent Notice

A useful trick is to make the first program line a REM line including a copyright notice, and after entering it to execute;

POKE 16510,0

This changes the line number to 0, and you will then find that the line can't be changed or deleted in the normal way. But of course anyone could simply change the line number back to something manageable by POKEing 16510 with another value, say 1.

A more confusing way of protecting your copyright notice, and one which lets you protect more than one line, is to proceed as follows;

- First, before entering anything else, enter the line or lines to be protected.
- Then enter POKE 16509,40 (actually you could POKE any value between 40 and 63). This should change the number of the first line to a letter followed by 3 digits.
- Then enter the rest of your program.

The lines present when you POKEd 16509 will then be visible at the end of the program, but cannot be changed or deleted in the normal way, and now there is no easy way of finding out which address has to be POKEd to change the line number back to a 'proper' one, as entering the main body of your program will have moved the POKEd byte up in memory.

Other BASICs

A wealth of programs written in 'BASIC' can be found in other books and in computer magazines. But sadly there are many versions of 'BASIC'. Although they may look similar, there are enough differences between the versions that a program written for another machine probably won't work on the ZX81 without some changes.

To help you find out what changes are needed, the following paragraphs list the main differences between ZX81 BASIC and the dialects used by other popular machines. But note that the extent and nature of the changes needed will depend on the structure of the program being converted, on how it handles data, and - most importantly - on the screen display characteristics of the particular machine it was intended to run on. So work out exactly how the program works before attempting to run it on the ZX81, it will save time in the long run.

Multiple-statement lines

Some BASICs allow multiple statements on a line, usually separated by ':' or '\' (the Acorn Atom uses ';') e.g;

```
10 LET A=B+C : LET D=2*D+A : PRINT A,D
```

These will have to be written on separate lines for the ZX81;

```
10 LET A=B+C
11 LET D=2*D+A
12 PRINT A,D
```

Variables

Most modern BASICs use floating point arithmetic and variables like the ZX81, but some also allow you to specify *integer* variables by adding a '%' to the variable name. Thus;

A would be a floating point variable.
A% would be an integer variable.

An integer variable is one which can only have integer (whole number) values. They are usually used because they take up less memory space than floating point variables, or because arithmetic using integer variables is usually faster, and in these cases there is no real harm in using the ZX81's floating point variables instead. But, occasionally, the program may rely on the properties of integer arithmetic, particularly integer division where the result is rounded down to give an integer result. If you suspect that this might be the case, then use the ZX81 INT function to 'integerise' the result of any division. E.g;

```
10 LET C%=Y /100
```

would become;

```
10 LET C=INT (Y/100)
```

A few BASICs, such as Apple II Integer BASIC, and 4K ZX80 BASIC, don't have floating point variables; all their variables are integer types even though they don't have the '%' symbol at the end of each variable name. Atom BASIC is even more confusing, as variables of the form 'A' are *integer* variables, while *floating point* variable names begin with '%', e.g. %A. Again, be wary of divisions.

Arrays

There should be no difficulty with numeric arrays, as the ZX81's capabilities are at least the equal of other machines. Note, however, that in most other BASICs subscripts start at zero, not at one. Thus DIM A(3) usually defines a four element array A(0) to A(3), whereas the ZX81 would create a three element array A(1) to A(3).

Some BASICs allow the use of multi-character variable names for arrays, these will have to be changed to single letter names for the ZX81.

The ZX81's string array facilities are also similar to those of most other machines (except that - again - most other BASICs start their subscripts at 0 rather than at 1). But there are differences in how they deal with the length of strings in an array. The DIM statement in most BASICs defines - as well as the size of the array - the *maximum* length of each string. Strings assigned to the array keep their original length unless they are too long, when an error report occurs. This is different to the ZX81's practice of adjusting the size of a string to make it fit exactly the string length DIMensioned for that array. The only real problem this might give is when comparing string variables, as discussed in 'DIM Strings' elsewhere in this book.

The following form of string array DIM statement may often be found;

```
DIM A$(2,2)[10]
```

This defines a 2x2 (or 3x3) string array, each element having a maximum length of 10 characters. It is equivalent to the ZX81's

```
DIM A$(2,2,10)
```
or
```
DIM A$(3,3,10)
```

Some BASICs require that every string variable - even those that are not array variables - be DIMensioned to reserve memory space for it. Other BASICs only require DIM statements for string variables which are likely to be longer than - say - 10 characters. In either case, you will encounter program lines such as;

```
DIM B$(10)    or    DIM C$[10]
```

which are not needed in the ZX81 version of the program.

Some BASICs allow more than one array to be DIMensioned by a single DIM statement, this cannot be done on the ZX81. Thus;

```
10 DIM A(2,2),B(50)
```

would have to be written as;

```
10 DIM A(2,2)
11 DIM B(50)
```

Arithmetic functions

Those used by the ZX81 are very similar to those available on other computers, except that some BASICs use the symbols '^' or '↑' instead of the ZX81's '**'.

Also, on some computers LOG is equivalent to the ZX81's LN, while others have both LN (logarithm to the base e ; as the ZX81's LN) and LOG (logarithm to the base 10).

ASC , CHR$

ASC(X$) returns the decimal value of the code for the first character in X$ and is therefore similar to the ZX81's CODE X$. Note, however, that most computers use the international standard ASCII code to represent characters, but the ZX81 is an exception. Thus ASC("A") gives a value of 65 on most machines, but CODE "A" on the ZX81 gives 38.

Apart from this coding difference, the ZX81's CHR$(X) function behaves the same as in other BASICs.

DATA , READ , RESTORE

These statements are included in many BASICs to allow a list of fixed values to be handled easily. DATA statements in a program establish the list, e.g;

```
100 DATA 7,2,5
110 DATA 12,13
```

then each READ statement takes the next value from the list, so;

```
200 FOR I=1 TO 5
210 READ A(I)
220 NEXT I
```

would load the values 7,2,5,12,13 into the array elements A(1) to A(5). RESTORE re-sets an internal 'pointer' to the start of the DATA list so that the next READ statement will get the first DATA item.

The most obvious way of simulating these statements in a ZX81 program is by a sequence of LET statements, e.g;

```
200 LET A(1)=7
201 LET A(2)=2
          etc.
```

Or - to save memory - you could store the values in the array by using direct LET commands (i.e. by entering the LET lines without line numbers). But don't forget that the associated DIM statements must have already been executed to set up the arrays, and when you have finished entering the values use GOTO rather than RUN to re-start the program or you'll lose all those laboriously entered values.

Using sliced strings is sometimes a better way of handling a lot of fixed data, as discussed in 'Saving Space'.

DEF FN

A statement of the form

```
10 DEF FNA(X)=X+X*X
```

is acceptable to most BASICs, and is taken to define a new function FNA(X) which will give the value X+X*X and can be used with different parameters during a program. Thus;

```
2Ø LET A=3*FNA(2)
3Ø LET B=FNA(A/2)
```

would give A the value 18, and B the value 90.

Usually up to 26 different functions - FNA() to FNZ() - can be defined.

One of the most annoying features of ZX81 BASIC is that it does not allow functions to be defined in this way.

We can write out the function in full each time it is encountered;

```
2Ø LET A=3*(2+2*2)
3Ø LET B=A/2+A*A/4
```

Or we could use a subroutine, but we will have to assign the correct value to each variable used by the subroutine before it is called, and re-assign the result on return;

```
2Ø LET Z=2
21 GOSUB 1ØØ
22 LET A=Z
3Ø LET Z=A/2
31 GOSUB 1ØØ
32 LET B=Z
---------
1ØØ LET Z=Z+Z*Z
1Ø1 RETURN
```

Alternatively, we could define a string variable as the required function, then execute it by a VAL function when required. But again we have to assign the correct value to each variable used in the function before it is used;

```
1Ø LET Z$="Z+Z*Z"
--------
2Ø LET Z=2
21 LET A=VAL Z$
3Ø LET Z=A/2
31 LET B=VAL Z$
```

END

Replace by STOP.

FOR··TO··STEP_____NEXT

Some BASICs allow FOR variables to have multi-character names. These will have to be changed to single character names for the ZX81.

A subtle problem can arise because many versions of BASIC always execute the statements between the FOR and NEXT at least once, even if the starting value of the control variable is greater than the terminating value. Thus;

```
10 FOR I=1 TO 0
20 PRINT "LINE 20"
30 NEXT I
```

will cause 'LINE 20' to be printed on some versions of BASIC, but not on the ZX81.

Some BASICs let you omit the variable name after the NEXT, they assume that you mean the one from the most recently used FOR statement. This is not allowed in ZX81 BASIC.

GET

Available in some BASICs, a statement of the form;

```
10 GET A$
```

waits for the user to press a key, then returns a single character string as A$.
INKEY$ is similar, but is a function rather than a statement, and does not wait for a key to be pressed. The ZX81 equivalent to the above line would be;

```
10 LET A$=INKEY$
11 IF A$="" THEN GOTO 10
```

GOTO, GOSUB

Some BASICs do not allow a 'computed' GOTO or GOSUB, such as the ZX81's GOTO A+100 . You may therefore be able to simplify a program slightly by taking advantage of this facility.

A few BASICs let you include a 'label' at the beginning of a line, immediately before or after the line number. This 'label' usually looks like a variable name, and may be followed by a semi-colon. The label can then be used after a GOTO or GOSUB instead of the target line number, e.g;

```
10 GOTO A
-------
100 A LET C=D
```

Which translates to;

```
10 GOTO 100
-------
100 LET C=D
```

Some BASICs have 'ON - - GOTO' and 'ON - - GOSUB' forms. These

make the program GOTO (or GOSUB) one of a number of lines, depending on the value of a variable. For example;

```
10 ON I GOTO 100,105,130
```

will jump to line 100 if I=1, to 105 if I=2 or to 130 if I=3.

Depending on the particular example, these forms may be replaced by a computed GOTO (or GOSUB) or by a sequence of IF - THEN - GOTO lines.

Or, you can be really tricky and use the AND operator, so the ZX81 version of this line would be;

```
10 GOTO (100 AND I=1)+(105 AND I=2)+(130 AND I=3)
```

IF··THEN

Most BASICs will allow you to write

```
IF - - THEN 100
```

instead of;

```
IF - - THEN GOTO 100
```

Some BASICs won't allow anything except a line number after the THEN. Programs written in these dialects may sometimes be simplified by taking advantage of the ZX81's ability to execute any statement following a THEN. Thus;

```
10 IF X>10 THEN GOTO 30
20 PRINT "TOO LOW"
30 ----
```

could be re-written as;

```
10 IF X<=10 THEN PRINT "TOO LOW"
30 ----
```

BASICs that allow multiple statements in one line usually also let you put multiple statements after a THEN, such as;

```
10 IF A>B THEN PRINT "A>B";LET A=B;GOTO 30
20 PRINT "A<=B"
30 ----
```

which would have to be re-written along the lines of;

```
10 IF A<=B THEN GOTO 20
11 PRINT "A>B"
12 LET A=B
13 GOTO 30
20 PRINT "A<=B"
30 ----
```

Note that BASICs differ in their interpretation of a *value* as being 'true' or 'false'. Thus;

```
10 IF A THEN PRINT "TRUE"
```

will print TRUE on a ZX81 if A is non-zero, whereas other versions of BASIC may recognise - for example - negative values as the Boolean 'false', and zero values as 'true'.

INPUT

Some forms of BASIC allow a 'prompt' string to be included in an INPUT statement. This string is printed out before the INPUT is performed, thus;

```
10 INPUT "NAME",N$
```

which would have to be re-written as;

```
10 PRINT "NAME"
11 INPUT N$
```

Also, some BASICs allow a single INPUT statement to get more than one value, e.g;

```
10 INPUT M,N
```

which becomes, for the ZX81 ;

```
10 INPUT M
11 INPUT N
```

LET

Some BASICs allow you to omit the word LET ;

```
10 A=B
```

is treated as if it were;

```
10 LET A=B
```

LEFT$, MID$, RIGHT$

These functions occur in many versions of BASIC. They operate on a string, returning a part of that string.

LEFT$(X$,Y) gives the first Y characters in X$, it's ZX81 equivalent is X$(TO Y)

RIGHT$(X$,Y) gives the rightmost Y characters from X$, and translates to X$(LEN X$-Y+1 TO)

MID$(X$,Y,Z) gives Z characters from X$, starting at the Yth. character of X$. It is equivalent to X$(Y TO Y+Z-1).

MOD, DIV

These are integer arithmetic functions;

DIV does an integer division, so

```
1Ø LET A=B DIV C
```

converts to;

```
1Ø LET A=INT (B/C)
```

MOD gives the remainder after an integer division, so;

```
1Ø LET A=B MOD C
```

would be, in ZX81 BASIC;

```
1Ø LET A=B-INT (B/C)
```

PEEK, POKE, USR, CALL, LINK, ?

LINK and CALL are similar to USR in that they call a machine language routine, but they do not return a value to the calling BASIC program. '?' is used in a highly exotic way by the Acorn Atom for both PEEK and POKE.

The effect of these commands depends entirely on the particular machine being used. To convert a program containing any of these to run on the ZX81 you must find out exactly what the command was intended to do, then write ZX81 code to perform a similar function.

PLOT

Each version of BASIC that has a plotting capability seems to use a different form of the PLOT statement. All you can do is find out what the original was intended to do, and write a ZX81 routine to do the same.

PRINT

PRINT statements will often have to be changed to make best use of the ZX81 screen format. Note that programs not written for the ZX81 may assume that the print-out is to be done on a display which automatically scrolls as required, and may need changing to prevent a display area overflow.

RND

Some BASICs need RND to be followed by a number enclosed in brackets, e.g; LET A=RND(1).

This number can be omitted in ZX81 BASIC, except where the program was written for a BASIC which handles only integer values (such as 4K ZX80 BASIC). In these cases, RND(n) usually gives a random integer value between 1 and n, or between 0 and n-1.

String concatenation

Some BASICs use the symbol '&' or '!' to join two strings, e.g;

```
LET A$=B$ & C$      or      LET A$=B$ ! C$
```

which are equivalent to the ZX81's;

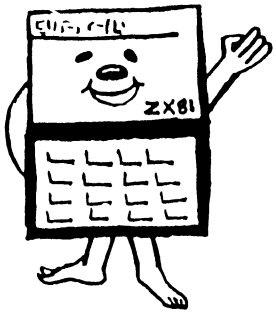
```
LAT A$=B$+C$
```

VAL

The ZX81 version of VAL is particularly annoying to use as it will stop the program unless the string if a completely valid numeric expression; it even objects to a leading + sign. Most other BASICs are more tolerant, and will just return the value zero if the string can't be evaluated.

Chapter 2

Machine Language



Binary & Hexadecimal

Why do people write Machine Language programs in Hexadecimal ? Why do computers work in Binary ? What is a Byte ?

If you already know the answers please feel free to skip this section. If not, then read on !

If you take a look at the ZX81 rear connector diagram shown in chapter 26 of the Sinclair ZX81 BASIC Manual, you should see that eight of the connections are labelled DØ - - D7. These form the computer's 'data bus', and carry all of the data being manipulated, and all of the instructions which say how to manipulate that data, between the Z80 processor and the ROM and RAM memories. When you plug the ZX81 printer onto this connector, it also uses the data bus lines to communicate with the processor.

When any device (RAM,Z80 etc,) is reading data from the data bus it looks for one of two voltage levels on each of the data lines DØ - D7;

- A 'low' level,which is anything below about 0.8 volts.
- A 'high' level, anything above about 2 volts.

And when a device is putting data onto the bus, it feeds each of the data lines with;

- A 'low' level of between 0 and 0.4V.

or - A 'high' level of between 2.4 and 5V.

Comparing the voltage levels sent onto the bus with those accepted by a device which is reading data from the bus lines, you can see that there is a reasonable margin and small amounts of electrical 'noise' which may get added onto the signal are ignored. This is good for the security of your data and one reason why digital computers have proved so popular. The other reason is that it is relatively easy - and hence cheap - to manufacture integrated circuits to meet the wide tolerances on the input threshold and output voltage levels.

The result of all this is that we have a data bus consisting of eight wires each carrying one of two voltage levels. If we represent the 'high' voltage level by the number '1', and the 'low' level by 'Ø', then we can write the combined signal on the data bus at any time as a string of eight '1's or 'Ø's e.g;

1Ø1ØØ11Ø

Now when we write down a number in our 'natural' ten-fingered decimal system, we use the symbols 0 to 9, and write the individual digits of a number in order so that - working from the right hand digit towards the left hand digit - each digit is assumed to be multiplied by the next power of 10. Thus

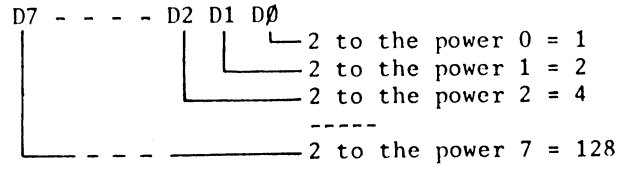
```
          1328

means       8 * 1
          + 2 * 10
          + 3 * 100
          + 1 * 1000
```

The eight signals on the data bus are treated similarly,except that as each data bus line, or digit of the number written, can only be one of two symbols (Ø or 1), then each digit in the number is assumed to be multiplied by the next power of 2.Thus;

```
          1Ø1ØØ11Ø

means       0 * 1
          + 1 * 2
          + 1 * 4
          + 0 * 8
          + 0 * 16
          + 1 * 32
          + 0 * 64
          + 1 * 128
```

Since we are working with 2 symbols, this is known as a 'binary' numbering system, as opposed to our 'natural' decimal (or denary) system which uses 10 symbols. Each digit in a binary number is known as a 'bit'.

Referring back to the data bus, DØ corresponds to the rightmost (least significant) binary digit, and D7 to the leftmost (most significant) one. You wil - of course - have noticed that the number of each data line corresponds to the relevant power of 2;



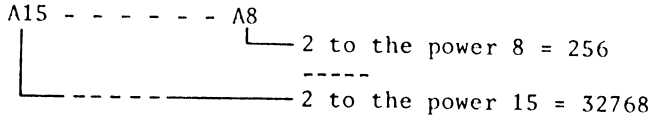
Which is convenient and - presumably - why the data lines are numbered Ø to 7 rather than 1 to 8.

The combined eight bits are known as a 'byte' - perhaps because the Z80 processor 'gobbles up' the data in these 8-bit chunks. For the record, a group of 4 bits is sometimes known as a 'nibble'.

If you care to work it out, you will find that a byte can represent any number in the range zero (ØØØØØØØØ) to 255 (11111111). Numbers in this range are used by the ZX81 to represent alphanumeric and graphics characters, keywords, and various other tokens as listed in Appendix A of the Sinclair ZX81 BASIC Manual.

Also on the ZX81's rear connector are sixteen lines labelled AØ to A15. These are the computer's 'address' lines, which again

carry binary signals. They are driven by the Z80 processor, and the number they carry specifies which memory byte the processor wants to read from or write to.The first eight lines AØ to A7 each correspond to the first eight powers of 2 - as with the data lines. The other eight address lines correspond to increasingly higher powers of 2;



In this way the binary signals on the 16-bit address bus can represent numbers from Ø to 65535, giving the ZX81 a theoretical capability of addressing up to 65536 memory bytes, or (using the computer enthusiast's shorthand 'K' meaning 1024) 64K bytes.

Hopefully it is now clear that when you are dealing with the signals on the ZX81's data and address busses you are really dealing with 8 or 16 bit binary numbers. Since a Machine Language program is concerned almost entirely with these 8 and 16 bit numbers, and often with the state of individual bits within a data byte, it makes sense for us to write such programs in a form which is close to the way the machine actually handles data and addresses. For example, in writing a data byte as;

1ØØØØØØ1

it is much easier to see whether - say - bit 3 is a Ø or a 1 than if we had written the equivalent decimal number 129.

But, binary numbers are not all that easy for humans to handle. For example, try to remember the 16-bit address;

11ØØ1Ø11ØØ111ØØØ

for five minutes. What is needed is some half-way house, which allows us to handle the numbers easily but without decimal's drawback of having no easy correspondence with the individual bits in the original binary number. This is where Hexadecimal makes its triumphant appearance !

The Hexadecimal numbering system works to the base 16 - rather than the base 10 used by decimal and base 2 by binary.
It therefore requires 16 different symbols to represent the digits, and for convenience the numbers Ø to 9 plus the letters A to F are used. Because there are sixteen possible symbols, each hexadecimal digit is exactly equivalent to four binary bits;

Binary	Hexadecimal	Decimal
ØØØØ	Ø	Ø
ØØØ1	1	1
ØØ1Ø	2	2
ØØ11	3	3
Ø1ØØ	4	4
Ø1Ø1	5	5
Ø11Ø	6	6
Ø111	7	7

Binary	Hexadecimal	Decimal
1ØØØ	8	8
1ØØ1	9	9
1Ø1Ø	A	1Ø
1Ø11	B	11
11ØØ	C	12
11Ø1	D	13
111Ø	E	14
1111	F	15

Having found a convenient way of representing four binary bits, all that remains is to divide our 8-bit byte or 16-bit address into 4-bit nibbles. Using the example given earlier;

```
11ØØ1Ø11ØØ111ØØØ
  C  B  3  8
```

Now isn't 'CB38' easier to remember than that long string of Ø's and 1's ?

Converting a hexadecimal number to decimal is also fairly easy as - working from right to left - each hexadecimal digit represents a successively higher power of 16. Thus;

```
CB38 ——    8*1          =     8
  |  └———  + 3*16        =    48
  |  └———  + 11*16*16    =  2816
  └———     + 12*16*16*16 = 49152
                           52024
```

Floating point binary

The ZX81 stores the values of numbers (variables and constants) in 5-byte binary floating point form.

The first of the five bytes is used to hold the binary 'exponent', the remaining four bytes hold the sign of the number and the 'mantissa'. The overall value of the number is;

'sign' 'mantissa' times 2 to the power 'exponent'

so if the sign was '+', the mantissa '3' and the exponent '4', the value of the number would be;

+ 3 times 2 to the power 4 = 48

In practice the mantissa is always scaled by multiplying or dividing it by powers of 2 , and the exponent adjusted accordly , so that the mantissa is a 32 bit binary fraction in the range .10000---- to .1111111---- (in decimal terms this would be .5 to .9999999--).

Because the left hand (most significant) bit of the mantissa is always a '1', there is no need to actually store it, so the ZX81 puts a 'sign' bit in its place. This is 1 for a negative number, Ø for a positive one.

The value of the exponent byte is offset by -128, so that;

ØØØØØØØØ	represents an exponent of -128
Ø1111111	represents an exponent of -1
1ØØØØØØØ	represents an exponent of Ø
11111111	represents an exponent of 127

A number having the value 'Ø' presents a slight problem, as you can't accurately represent it by a mantissa having a leading '1'. In practice the ZX81 stores the value 'Ø' as five zero bytes, or .5 times 2 to the power -128, but it takes care in arithmetic routines to remember that this actually represents zero.

Using USR

The USR function allows us to call a Machine Language routine,and can return a value to the calling BASIC program.

The Sinclair ZX81 BASIC Manual is rather reticent about its use, but perhaps that is to be expected, as USR opens the door to the whole complex and detailed topic of Machine Language programming which is a subject large enough to justify a book in its own right. In fact several have been written, of which possibly the best for the ZX81 owner is 'Mastering Machine Code on your ZX81' by Toni Baker.

Because the subject is so large, we can't explore it fully in a general book like this either. What we can do, however, is to give the 'flavour' of Machine Language, with examples of simple Machine Code routines, so that the reader can see if it is a subject he wishes to study further.

The Z80 Processor

This is the 'brains' of the ZX81, where all the calculations are actually performed. It is connected to the ZX81's ROM and RAM memories, from where it can get instructions and data, and can put results back into the ZX81's RAM. It also has several small internal memories, known as 'registers', which it uses as a 'scratch pad' for holding temporary values. These registers each hold one 8-bit byte, or may be used in pairs to handle 16 bit words.

Although it is a very clever chip, the Z80 processor doesn't itself understand BASIC. What it does understand are programs written in a special language known as 'Z80 Machine Language' and stored in 'Machine Code'.

The ROM chip in the ZX81 contains such a machine language program, known as the 'BASIC Interpreter'. It is by running this program that the Z80 manages to understand and so execute your BASIC program. For each line in the BASIC program, the Z80 may actually have to execute hundreds, or even thousands, of instructions from the Interpreter program, doing things such as working out exactly what your statement means, checking for any errors, and calculating where the variables have been stored, as well as evaluating any string or arithmetic expression you may have used.

Other machine language routines within the ZX81's ROM are used for scanning the keyboard, for LOADing and SAVEing your BASIC program, and for providing the TV display.

USR may be used to call one of these routines within the ROM, or it may call a routine that has been specially written for some particular application and stored in an area of RAM.

Machine Language

Each instruction in Z80 Machine Language consists of one or more 8-bit bytes, which for convenience we usually write in hexadecimal rather than binary form, and comprises an Operation Code possibly followed by data or an address. Thus;

```
3C
```

means 'add one to the A register', and;

```
3E Ø1
```

means 'load the A register with the value Ø1'.

Since these hexadecimal codes are not all that easy to work with, machine language programs are usually first written in a mnemonic form which is later translated to hexadecimal. The examples shown above would be written in mnemonic form as;

```
INC A
```

and

```
LDA Ø1
```

Programs written in this mnemonic form are often known as 'Assembly Language Programs', and 'Assembler' programs are available that will take a program written in Assembly Language and convert it to hexadecimal and also to the actual binary codes used by the Z80 processor.

When giving a listing of a Machine / Assembly Language program, it is usual to show it in the following format;

- First the address of the memory location in which the first byte of the instruction is to be stored, expressed as four hex digits.
- Then the hexadecimal version of the instruction.
- Any 'label' used, followed by a colon.
- The mnemonic version of the instruction.
- Any comments you may care to put, explaining the routine.

An example, taken from the start of the ZX81's ROM, is;

address	instruction	label	instruction	comment
ØØØØ	D3 FD	RESET:	OUT FD,A	
ØØØ2	Ø1 FF 7F		LD BC, 7FFF	HIGH RAM ADDRESS

The 'labels' are used as convenient pointers to useful parts of the program, for example a later JUMP instruction could be written as;

JP RESET

rather than JP ØØØØ

The Assembler program will then fill in the correct address as it converts the mnemonic code to hex and binary.

In summary, the advantages of machine language routines are;

- They can be very fast, particularly in applications such as altering all or part of the display, searching for a particular sequence of codes in a large area of memory, or repeating a simple operation many many times. All of the best moving graphics games and those such as Chess which involve a lot of computation are written in machine code.
- They often occupy fewer memory bytes than a similar BASIC routine.
- They can be used to control ZX81 features (such as the tape in and out ports) which are not directly accessible from ZX81 BASIC.

But of course there are disadvantages, the main one being the length of time it takes to write and successfully debug a machine language program. If the author's experience is anything to go by it can take between ten and twenty times longer to get a machine language program working than it would have taken for a similar BASIC routine !

Where To Put It

The first problem to arise when you start to write a machine language program is where to put it. The machine code must be

stored somewhere in RAM, but must not conflict with ZX81 BASIC's use of memory. There are several alternatives.

REM at first

One way, as suggested in the Sinclair ZX81 BASIC manual, is to make the first line of your program a REM statement, with at least one character after the REM for each byte in your machine code, e.g;

```
1 REM 123456789Ø
```

which will reserve ten bytes (at locations 16514 to 16523) into which you can POKE the values of the machine code bytes.

The resulting BASIC program may look strange when you LIST it, but it can be SAVEd - complete with the POKEd machine code - and will RUN without any problems except that sometimes the value 118 (hex 76) can confuse the ZX81, as this is the code for NEWLINE. The value 126 (hex 7E) also needs some care as it makes the ZX81 BASIC interpreter skip the next 5 bytes - thinking that they are a floating point binary number. This stops it LISTing these five bytes, and if they include the start of the next line then the program will not run properly.

Once the machine code has been stored, the function USR x should be used to run it, x being the starting address of the machine code routine.

As a very simple example, enter the program;

```
1 REM 1234
2 PRINT USR 16514
```

then, before you RUN it, enter;

```
POKE 16514,1
POKE 16515,Ø
POKE 16516,Ø
POKE 16517,2Ø1
```

RUN should now give the result 'Ø' as the machine code routine was;

```
LD BC , ØØØØ
RET
```

i.e. load the register pair BC with the value Ø , then return to the calling BASIC program. As the function USR always gives the value of the BC register pair as a result, this is printed out.

Instead of laboriously POKEing every value, we can choose the characters typed after the REM to have the correct codes. Since;

```
CODE " " = Ø
CODE '■" = 1
CODE TAN = 2Ø1
```

we could enter the 4-byte machine code routine by simply keying in the line;

1 REM ■ *space space TAN*

noting that the symbol ■ is entered with the aid of the GRAPHICS key, and TAN with the FUNCTION key.

Looking through Appendix A of the Sinclair ZX81 BASIC Manual, we find that most codes can be stored this way, the exceptions being those in the range 67 to 127,and 195, as these are either unused or are the editing keys ,FUNCTION,DELETE etc. A slight problem arises with the *Statement* keys having codes 230 to 255 (NEW SCROLL etc.) as the ZX81's screen editor does not normally allow these to be input after a REM. Luckily it is easily fooled, just enter the line from right to left, using the ← key to re-position the cursor after each keystroke.

REM at last

In the previous section it was suggested that machine code could be incorporated in a REM statement as the first line of a program. In fact there is no reason why the REM statement should not appear anywhere in the BASIC program except that by putting it as the first line it always occupies the same position in memory, so the machine code always starts at location 16514. If we were to put it anywhere else then we would have to find out exactly where it was in RAM before we could use the machine code routine and any changes to previous lines would probably move it.

However, it may sometimes be a good idea to make the REM line the *last* line in the BASIC program. As the last line, it is not too difficult to find out where it is in RAM by working backwards from

```
PEEK 16396 + 256 * PEEK 16397
```

which gives the starting address of the Display File - which follows immediately after the end of the program area of RAM. The advantage of using the last line is that by POKEing a value of 255 into the first byte of the REM line (the most significant byte of the line number), the line won't be LISTed and the ZX81 won't attempt to RUN it as part of the BASIC program - thus allowing you to use the 'difficult' codes 118 and 126 with impunity, and also protecting it from prying eyes.Although the line is now 'invisible', you will still be able to SAVE and LOAD it along with the rest of the BASIC program.

Quote it

A machine code routine may also be held between quotation marks, as shown below;

```
100 PRINT "  machine code bytes  "
```

and the code POKEd or keyed directly as described for a REM line. But take care that the code for a quotation mark (11 , 0B hex) is not included, and of course the line should not be *executed* by the BASIC program. Perhaps the best place to put such a line would be at the end of the program, where it could be made "invisible" , but a REM line is just as good and 2 bytes shorter.

With the Variables

Machine Code may also be stored in the Variable Area of RAM, in space reserved by DIMensioning a suitably sized single dimensioned character or numeric array, e.g;

```
10 DIM A$(50)     or     10 DIM A(10)
```

which will both reserve space for 50 bytes of machine code.

It is best to make the DIM statement which reserves the space the first line of your program to ensure that the array is placed right at the start of the Variable Area where it can be easily located.

Machine code stored in this area will be SAVEd along with your program, but remember to use GOTO rather than RUN, and not to re-execute the DIM statement if you don't want to lose the stored code.

The main disadvantage of this method of storing machine code is that the Variable Area gets moved around in RAM as the program and the Display File are changed, so you have to be careful in calculating where the machine code is. The main advantage is that there is no restriction on the codes that can be used.

The codes will usually be entered into the array space by POKEing, although there is no reason why direct assignment lines shouldn't be used - particularly if a string array is being used, as for example;

```
LET A$(2) = CHR$ 222
```

or even;

```
LET A$(2 TO 4) = "A-C"
```

In high RAM

As described in Chapter 26 of the Sinclair ZX81 BASIC Manual, you can reserve space at the top of RAM for a machine code routine by POKEing a new value into the System Variable RAMTOP (locations 16388,16389) and then executing a NEW command. All

of the RAM above the new value of RAMTOP is then free for you to use as you wish. But note that the new value of RAMTOP remains until you either POKE a new value or turn the machine off, but - dissapointingly - it is not saved on tape so that every time you want to use a program which takes advantage of this facility you must POKE RAMTOP before LOADing the program.

This area is a good place for machine code because it doesn't interfere with the BASIC program in any way, and once placed there, your code is in a fixed location. The main drawback, however, is that this area of RAM is not SAVEd to tape.

If you have enough memory available, then it is possible to overcome the SAVEing problem by copying the machine code into an array before SAVEing, then copying it back to the top of RAM after loading.

LPRINT Buffer

If you're not using the ZX printer, then the System Variables PRBUFF area (16444 to 16476, hex 403C to 405E) can hold up to 33 bytes of machine code, and the locations 16440, 16507 and 16508 can be used for holding 8 or 16 bit variables.

Unfortunately, this area is cleared whenever a direct command (e.g. RUN or GOTO) is executed.

Hex it

Until now, we have talked in terms of POKEing the machine code bytes directly into RAM, or entering them as their character or keyword equivalents. Both methods make it very difficult to see what you have actually done, and so complicate debugging of the machine code routine.

If you have enough memory, then it is often better to include the hexadecimal form of the machine code in a string or strings within the BASIC program, and have a routine which will translate the hexadecimal into binary bytes and POKE them into the correct memory locations. Because the machine code is now visible in hexadecimal form, it is easy to check it and to make any necessary changes.

Of course we still have to decide where in RAM the machine code proper is to go, but since the hexadecimal equivalent is now able to be SAVEd along with the rest of the BASIC program, the area above RAMTOP is a good choice. As an example of this technique, we can take the short routine used previously;

```
Ø1 ØØ ØØ  LD BC, ØØØØ
C9         RET
```

First, enter;

```
POKE 16388,252
POKE 16389,67
NEW
```

to reserve 4 bytes at locations 174Ø4,5,6,7 (more bytes will be reserved if you have more than 1K of RAM).

Then enter and RUN the following program;

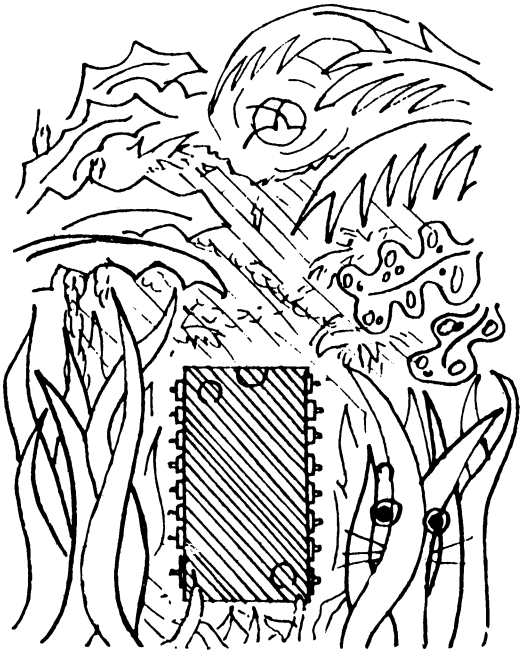
```
1Ø LET A=174Ø4
2Ø LET A$="Ø1ØØØØC9"
3Ø FOR I=1 TO 7 STEP 2
4Ø POKE A,16*CODE A$(I)+CODE A$(I+1)-476
5Ø LET A=A+1
6Ø NEXT I
7Ø PRINT USR 174Ø4
```

Again, it should set the Z80's BC register pair to zero then print out that value.

For real

Some of the programs given in chapters 4 and 6 (such as Alien Attack, All Change, and Catalogue) use specially written Machine Code routines. These can either be keyed into the ZX81 without worrying how they work, or can be studied to get a better understanding of the 'flavour' of Machine Code.

Chapter 3



Discovering The ROM

ROM Routines

The ZX81's 8K ROM contains many routines which are useful to the machine language programmer. The following paragraphs give the starting addresses of the more important ones and a brief description of their function. All addresses and values are given in hexadecimal, and are for the new - 'correct' - ROM , which agrees that SQR 0.25 is 0.5. (The 'Old' ROM which was fitted in the first ZX81's had a couple of mistakes in it, the worst of which was a peculiarity in the arithmetic routines.)

0000 RESET
The ZX81's starting point on power-up. Looks to see how much RAM is fitted - and fills it with zeroes. Sets RAMTOP (4004, 5) to one more than the highest actual RAM address, then goes into the NEW routine. Called by JP 0000 or RST 0.

0008 REPORT
The ZX81's error report routine and is called by RST 08. The byte following the RST instruction is copied to ERR-NR (4000) and will cause an error halt if it is anything other than FF.

0010 PRTCHA
Called by RST 10, prints the character whose code is in the Z80's A register at the current print position. The major portion of the code for the print routines lies from 07F1 and 094A.

0018 GTCHA
Called by RST 18, returns with the A register containing the byte (from your program) pointed to by CH-ADD (4016). Skips spaces - incrementing CH-ADD - and the cursor.

0020 GNXCH
Called by RST 20, as GTCHA but skips the character that CH-ADD is currently pointing to.

0030 MOREV
Called by RST 30, increases the variable area by the number of bytes given by the BC register pair.

0038 INT
Maskable Interrupt Handler routine, called by a hardware interrupt for each TV line when the ZX81 is generating the display. See 'Display'.

0066 NMI
Non-maskable Interrupt Handler routine, called by a hardware interrupt each TV frame when in the SLOW (compute and display) mode to change between the compute and the display modes. See 'Display'.

ØlFC NXTBYT
Used by the LOAD & SAVE routines to increment HL to point to the next byte to be LOADed or SAVEd. Jumps to the Display routine at 0207 when HL equals E-LINE.

Ø2Ø7 DISP
The start of the display routines, see 'Display' for full details.

Ø2BB KSCAN
The keyboard scan routine, see 'Display' for full details.

Ø2F4 SAVE
The ZX81's SAVE routine; see 'Recording'.

Ø34Ø LOAD
The ZX81's LOAD routine, see 'Recording' for details.

Ø7Fl OUTCHA
Sends the character whose code is in the Z80's A register to the display if bit 1 of FLAGS (4001) is 0, or to the printer if bit 1 of FLAGS is 1. Uses DISPA (0808) or PNTA (0851).

Ø8Ø8 DISPA
Puts the character whose code is in the A register to the current print position on the TV screen, then updates S-POSN (4039,A) and DF-CC (400E) to point to the next print position.

Ø85l PNTA
Sends the character whose code is in the A register to the printer line buffer PR-BUFF at the position given by PR-CC (4037). If the buffer then holds a complete line, or if the character was NEWLINE (76 hex) the buffer is printed out. PR-CC is then updated.

ØA2A CLS
Performs exactly as the BASIC CLS command.

ØAAl PHL
Prints the contents of the HL register pair as a decimal number (0 to 9999).

ØAAB PHLS
Prints the contents of the HL register pair as a 4-digit decimal number 0-9999 with any leading spaces being printed as spaces.

ØB6B PSTR
Prints a string. DE should point to the first character in the string, BC should contain the number of characters in the string. The routine will recognise keyword codes (e.g. F8 hex for SAVE) and print them out in their full form.

ØF23 FAST
Change to Fast mode.

ØF2B SLOW
Change to Slow mode.

174C FSUB
Subtracts the 5-byte floating point binary number at (DE) from the 5-byte floating point binary number at (HL).

1755 FADD
Adds the 5-byte floating point binary number at (DE) to the 5-byte floating point binary number at (HL).

17C6 FMLT
Multiplies the 5-byte floating point binary number at (HL) by the 5-byte floating point number at (DE).

1882 FDIV
Divides the 5-byte floating point number at (HL) by the 5-byte floating point binary number at (DE).

ROM Tables

The ZX81 ROM also contains several useful tables, which are briefly described in the following paragraphs. Again, the addresses are given in hexadecimal and are for the new 8K ROM.

ØØ7E - Ø11Ø KEY TABLES

The ZX81 keyboard scanning routines return a 'key number' (1 to 39) if a valid keypress is detected. These Key Tables are used to translate the key number into the appropriate code.

007E to 00A4 is used for the unshifted L mode, e.g. the letters, numbers, '.', space and NEWLINE.

00A5 to 00CB is used for the shifted L mode, e.g. STEP, TO, ')' etc.

00CC to 00F2 is used in the F mode, e.g. for LN, RND etc.

00F3 to 0110 is used in the shifted G mode, for the graphics characters and some inverse symbols.

Ø111 - Ø1FA KEYWORD TABLE

Codes 192 to 255 and 64,65,66 are displayed as more than one character (e.g. SLOW or RETURN). This table holds the 'spelled out' versions of these codes. They are in code order, from codes 192 to 255 then 64,65 and finally 66. The last character of each keyword has bit 7 set to 1.

1EØØ - 1FFF DOT PATTERN TABLE

Holds the dot patterns for each of the codes 0 to 64, eight bytes being used for each code, corresponding to the eight scan lines used to display each character.

LOAD & SAVE

When the ZX81 SAVEs a program to tape, it writes;

- A 5 second silent period.
- The 'name' of the program, exactly as you entered, one byte per character, with the last character inverted (with bit 7 set to 1).
- The contents of RAM from location 16393 (4009 hex) - the System Variable VERSN - to the end of the Variable Area, which is one less than the address in the System Variable E-LINE.

Each byte is sent to the recorder as eight bits - no start, stop or parity bits are used - with the most significant bit 7 being sent first.

Each bit is recorded as 4 cycles (for a '0') or 9 cycles (for a '1') of approximately 300Hz, followed by a 1.5millisecond pause.

When LOADing, the ZX81 first looks for the silent period, then checks the name - recognising the last character of the name as being the first byte with bit 7 set to 1. It then loads subsequent bytes received from the tape into successive RAM locations starting at 16393 (4009 hex) until it reaches the address which is equal to the just loaded value of the System Variable E-LINE. It then goes to the display routine.

The SAVE Routine

The starting address for the main SAVE routine is 02F6.

It uses the subroutine at 03A8 to put the address of the first byte of the name the program is to be saved under into the DE register pair and set bit 7 of the last byte of the name to 1. The subroutine returns with the C flag set to 1 if no name was given.

The second subroutine called - at 0F46 - returns with the C flag set to zero is the BREAK key is pressed and so halts the SAVE.

The routine at 01FC - which is also used by the LOAD routine - increments HL and terminates the SAVE (or LOAD) by jumping into the Display routine when HL reaches the value of the System Variable E-LINE.

```
Ø2F4  CF ØE          RST 8
Ø2F6  CD A8 Ø3       CALL Ø3A8      Check name,
Ø2F9  38 F9          JR C Ø2F4       quit if no good.
Ø2FB  EEB            EX DE HL
Ø2FC  11 CB 12       LD DE 12CB
Ø2FF  CD 46 ØF       CALL ØF46      | 5 second
Ø3Ø2  3Ø 2E          JR NC Ø332     | silent
Ø3Ø4  1Ø FE          DJNZ Ø3Ø2      | period.
Ø3Ø6  1B             DEC DE         } Quit if
Ø3Ø7  7A             LD A D         | BREAK key
Ø3Ø8  B3             OR E           | pressed.
Ø3Ø9  2Ø F4          JR NZ Ø2FF     |
Ø3ØB  CD 1E Ø3       CALL Ø31E      | Write name
Ø3ØE  CB 7E          BIT 7 (HL)     } to tape,
Ø31Ø  23             INC HL         | last char has
Ø311  28 F8          JR Z Ø3ØB      | bit 7 set to '1'
Ø313  21 Ø9 4Ø       LD HL 4ØØ9     | Write RAM contents
Ø316  CD 1E Ø3       CALL Ø31E      } from 4009
Ø319  CD FC Ø1       CALL Ø1FC      | to E-Line - 1
Ø31C  18 F8          JR Ø316        | to tape.

Ø31E  5E             LD E (HL)      Get next
Ø31F  37             SCF             byte.
Ø32Ø  CB 13          RL E           Quit if
Ø322  C8             RET Z           last bit
Ø323  9F             SBC A A          makes A equal;
Ø324  E6 Ø5          AND Ø5            4 if bit is Ø
Ø326  C6 Ø4          ADD A Ø4          9 if bit is 1
Ø328  4F             LD C A         Send 4 or 9 pulses.
Ø329  D3 FF          OUT (FF) A     Send
Ø32B  Ø6 23          LD B 23         one
Ø32D  1Ø FE          DJNZ Ø32D       pulse.
Ø32F  CD 46 ØF       CALL ØF46      Quit if
Ø332  3Ø 72          JR NC Ø3A6      Break
Ø334  Ø6 1E          LD B 1E        Pause
Ø336  1Ø FE          DJNZ Ø336       between
Ø338  ØD             DEC C            pulses.
Ø339  2Ø EE          JR NZ Ø329
Ø33B  A7             AND A          Pause
Ø33C  1Ø FD          DJNZ Ø33B       between bits
Ø33E  18 EØ          JR Ø32Ø        To next bit.


Ø1FC  23             INC HL         Point
Ø1FD  EB             EX HL DE        HL to
Ø1FE  2A 14 4Ø       LD HL (4Ø14)     next byte
Ø2Ø1  37             SCF               then
Ø2Ø2  ED 52          SBC HL DE          return
Ø2Ø4  EB             EX HL DE             if not yet
Ø2Ø5  DØ             RET NC                 equal to E-LINE
Ø2Ø6  E1             POP HL         Clear stack.
                                    Go to Display routine.
```

The LOAD Routine

This starts at 0340 and immediately calls the subroutine at 03A8 which points DE to the start of the program name and sets bit 7 of the last byte of the name to '1'. If no name has been given then the C flag is set to '1' and the instructions RLD RRCD set bit 7 of the D register also to '1'.

The next two instructions; CALL 034C JR 0347 wait for the silent period at the start of each recording. This is done by calling the byte input routine at 034C which will always return (via the RET instruction at 03A1) as long as there are no breaks in the input signal lasting for more than about 5 mS.

This byte input routine is in two parts; the part from 034C to 035D waits for a signal on the tape input port - which causes it to jump to the second part of the routine at 0385 - or for the BREAK key to be pressed - which causes a jump to 03A2 to terminate the LOAD. If no signal is detected on the tape input port within about 5mS, then the program moves on to the routine starting at 035F.

The second part of the byte input routine, from 0385 to 03A1, measures the length of the signal burst coming from the cassette recorder - using the register E as a counter - and so decides if a '1' or a '0' has been received. It then stores the corresponding bit in the C register. When 8 bits have been received so that the C register is holding a complete byte, the routine returns to the calling program, if less than 8 bits have been received it jumps back to 034E to wait for the next bit from the tape.

The remainder of the LOAD routine - 035F to 0384 - first clears the return stack. It then quits, by jumping into the NEW routine at 03E5, if the register D contains zero. This can only happen if there is a loss of signal lasting for more than about 5mS after the ZX81 has found the correct program and started to load it into RAM.
If all is well, it reads the first bytes from the tape, matching them against the given program name. If there is a discrepancy, then the routine jumps back to 0347 to start all over again. If the name matches, then after recognising the last byte of the name as having bit 7 set to 1, the routine moves on to load in the program proper, starting at 0375.

Since the System Variable E-LINE is used as the limit to the LOAD process, its high byte is first incremented to make sure that at least the System Variables are read in - it will later be over-written with the correct value read from the tape.

HL is then pointed at the first RAM location to be loaded (4009) then the data is read from the tape a byte at a time and stored in consecutive RAM locations. The routine at 01FC is used each time to increment the HL pointer and terminate the LOAD when HL is equal to E-LINE.

```
034Ø  CD A8 Ø3          CALL Ø3A8     C is 1 if no name.
Ø343  CB 12             RL D          Make bit 7 of D '1'
Ø345  CB ØA             RRC D          if no name given.
Ø347  CD 4C Ø3          CALL Ø34C     Wait for silent
Ø34A  18 FB             JR Ø347        period.
Ø34C  ØE Ø1             LD C Ø1       Start of 'read a
Ø34E  Ø6 ØØ             LD B ØØ        byte' routine.
Ø35Ø  3E 7F             LD A 7F
Ø352  DB FE             IN A (FE)     Flash
Ø354  D3 FF             OUT (FF) A     TV, check
Ø356  1F                RRA             keyboard.
Ø357  3Ø 49             JR NC Ø3A2    Quit if Break.
Ø359  17                RLA           Goto 0385
Ø35A  17                RLA            if cassette
Ø35B  38 28             JR C Ø385       input found.
Ø35D  1Ø F1             DJNZ Ø35Ø     Back for next bit.
Ø35F  F1                POP AF        Quit if silence for
Ø36Ø  BA                CP D           longer than 5mS.
Ø361  D2 E5 Ø3          JP NC Ø3E5
Ø364  62                LD H D        Point HL to start
Ø365  6B                LD L E         of name.
Ø366  CD 4C Ø3          CALL Ø34C     Byte from tape in C.
Ø369  CB 7A             BIT 7 D       Jump if
Ø36B  79                LD A C         no name
Ø36C  2Ø Ø3             JR NZ Ø371      was given.
Ø36E  BE                CP (HL)       Check name, try again
Ø36F  2Ø D6             JR NZ Ø347    if wrong.
Ø371  23                INC HL
Ø372  17                RLA
Ø373  3Ø F1             JR NC Ø366    End if bit 7 is '1'
Ø375  FD 34 15          INC (IY+15)
Ø378  21 Ø9 4Ø          LD HL 4ØØ9    Start of RAM
Ø37B  5Ø                LD D B         to be loaded.
Ø37C  CD 4C Ø3          CALL Ø34C     Get next byte
Ø37F  71                LD (HL) C      save it
Ø38Ø  CD FC Ø1          CALL Ø1FC       enough ?
Ø383  18 F6             JR Ø37B       Go get another.

Ø385  D5                PUSH DE       Start to get
Ø386  1E 94             LD E 94        byte from tape.
Ø388  Ø6 1A             LD B 1A
Ø38A  1D                DEC E         Decr E while
Ø38B  DB FE             IN A (FE)      tone burst
Ø38D  17                RLA             being
Ø38E  CB 7B             BIT 7 E           received
Ø39Ø  7B                LD A E
Ø391  38 F5             JR C Ø338     Jmp if cassette i/p
Ø393  1Ø F5             DJNZ Ø38A
Ø395  D1                POP DE
Ø396  2Ø Ø4             JR NZ Ø39C    Put bit
Ø398  FE 56             CP 56          in C register.
Ø39A  3Ø B2             JR NC Ø34E    Jump to 034E
Ø39C  3F                CCF            after bits 0-6,
Ø39D  CB 11             RL C          Return after
Ø39F  3Ø AD             JR NC Ø34E     seventh bit
Ø3A1  C9                RET
```

Display

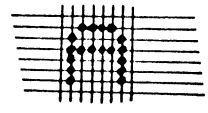
One of the outstanding features of the ZX81 is the way the TV display is generated with the minimum amount of hardware. This as much as anything else allows Sinclair to sell the machine at such a low price. Since many sophisticated ZX81 programs are based on the clever use of the display facilities, it is worth investigating them in some depth.

On the TV

The image on the TV screen is made up from several hundred horizontal lines drawn by a rapidly moving dot. Each line takes 64 micro-seconds to draw, and a complete 'frame' or screen full of lines is drawn every 1/50 th. of a second. (1/60 th. of a second for U.S.A. models). As the dot moves across the screen, it is rapidly turned on and off by the signal from the ZX81 to build up the black and white picture.

To make sure that the picture appears at the right place on the screen, the signal from the ZX81 also includes 'line synchronisation' pulses at the start of each scan line, and 'frame synchronisation' pulses at the end of each complete frame.

Each of the 24 lines of characters which can be displayed on the screen consists of 8 horizontal TV scan lines, and each character position on the screen is made up of 8 possible 'dots' in each of the 8 scan lines;



The dot patterns for each displayable character are stored in the ZX81's ROM, one byte of the ROM being used for each of the 8 TV scan lines for each character. To create the display, the ZX81 has to read the required bytes out of the ROM at the correct times, and use them to generate a sequence of 'black' and 'white' signals to send to the television set.

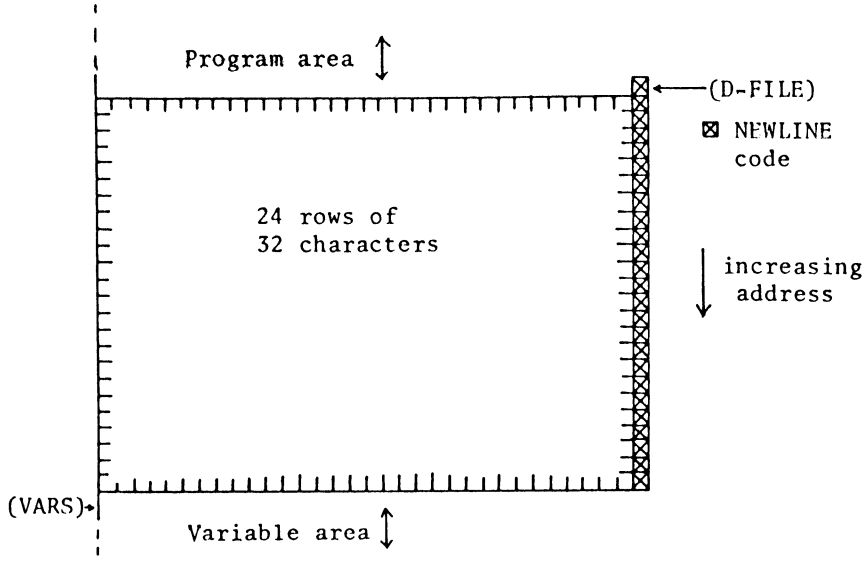
The Display File

This is the area of RAM that holds a 'copy' of whatever is to be displayed on the screen. If your ZX81 has 4K bytes or more of RAM then the Display File will consist of 24 'lines' each containing the 32 characters (including spaces) to be shown, plus a NEWLINE character (CODE 118, hex 76) . A 25th. NEWLINE character is also present at the start of the Display File.

The System Variable D-FILE (16396/7, hex 400C/D) contains the address of the start of the Display File - actually the address

of the first (NEWLINE) character, and DF-CC (16398/9, 400E/F) contains the RAM address corresponding to the current print position, that is, the address of the byte in the Display File which will contain the next character to be printed. Since the Variables Area follows immediately after the Display File, the Syatem Variable VARS (16400/1, 4010/1) contains the address of the RAM byte immediately following the last byte of the Display File.

We can represent the area of RAM taken by the Display File on a ZX81 with 4K or more RAM by a 'screen image' drawing such as;

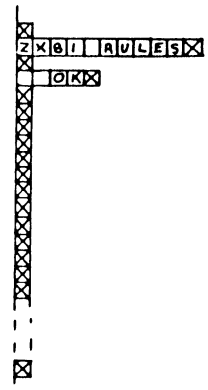


Note however that - even with 4K or more of RAM - SCROLL reduces the display file size by making the new - bottom - line consist of just a single NEWLINE character.

If you have less than 4K RAM, then NEW, LOAD or CLS will 'collapse' the Display File to the minimum of 25 NEWLINE characters, and subsequent PRINT or PLOT statements will only expand it enough to fit in the printed or plotted item. For example;

```
10 CLS
20 PRINT "ZX81 RULES"
30 PRINT AT 2,2;"OK"
```

will result in a Display File looking like;

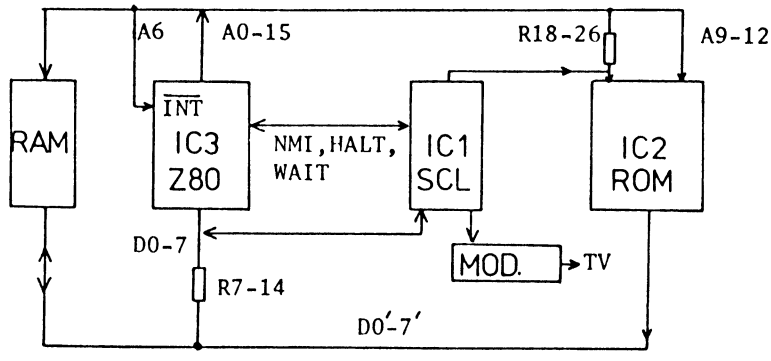


Note that although keywords such as REM or THEN are stored as a single byte in the Program Area of RAM, they are expanded to the correct number of characters in the Display File.

Note also that bits 0 to 5 of each character in the Display File define which character or graphics element is to be displayed, bit 7 is normally 0 but is set to 1 for an inverse (white on black) character or graphics symbol, and bit 6 is 0 except for in the NEWLINE code.

The Hardware

The major circuit elements involved in producing the display are;



When no display is being generated, i.e. when the ZX81 is running your program in FAST mode, the ZX81 functions as a normal microcomputer controlled by the Z80 processor IC3, and the series resistors R7-14 and R18-26 in the data lines and in some address lines have no effect.

However, these series resistors do allow the SCL chip IC1 to 'force' data onto the Z80's data input lines D0-D7 regardless of the current output of the ROM and RAM chips, and similarly IC1 can force the nine least significant address lines A0-A8 to any required state regardless of the address being output by the Z80.

Note also that when the Z80's Maskable Interrupt is enabled by suitable software then any address which has A6=0 put out by the Z80 processor during the end of an instruction cycle will cause an interrupt.

The Z80 processor has the unique ability to generate 'refresh' addresses for use with dynamic RAM chips. After each machine code instruction has been fetched from ROM or RAM, the Z80 puts the contents of its 7-bit internal 'refresh' register R onto the address lines A0-A6 and pulls the RFSH line low while it is decoding the instruction. When it is ready to proceed, the Z80 takes the RFSH line high again and increments the R register. This feature was intended to ease the use of dynamic memories, but has been used in the ZX81 as the heart of the display mechanism, since certain instructions cause the refresh address to be put out at the time that the Z80 is looking at the Maskable Interrupt input.

The final clues to how the ZX81 generates the display are;

- The SCL chip can detect that the Z80 is fetching an instruction from an address in the top half of memory space, i.e. when A15 is at '1'.
- The NEWLINE code (76 hex) is the Z80's HALT code, and the only code used in the Display File which has bit 6 equal to a '1'.
- When the Z80 is halted, it executes NOP (No Operation) instructions repeatedly while waiting for an interrupt.
- When the contents of the refresh register R are put onto the low 7 address lines, the Z80 also puts out the contents of its 8-bit register I onto the high address lines A8-15.

INT Handler

The maskable interrupt routine from 0038 to 0048 is the heart of the ZX81's display mechanism. It is;

```
ØØ38  0D        INT: DEC C        Next scan line
ØØ39  C2 45 ØØ       JP NZ ØØ45
ØØ3C  E1              POP HL      Get next char's address
ØØ3D  Ø5              DEC B        Next character row
ØØ3E  C8              RET Z       Ret if last row finished.
ØØ3F  CB D9           SET 3 C     Make C equal 8
ØØ41  ED 4F           LD R A      Reset refresh counter.
ØØ43  FB              EI          Enable Interrupt.
ØØ44  E9              JP (HL)     Display next scan line.
ØØ45  D1              POP DE      Clear stack
ØØ46  C8              RET Z
ØØ47  18 F8           JR ØØ41
```

If the Maskable Interrupt has been enabled, then this routine is automatically called when the Z80's INT line is pulled low which, as we shall see, happens at the start of each horizontal scan line during the display.

When the routine is called;

- The Z80's C register contains the number of scan lines left for the current row of characters.
- The stack contains the address of the next character in the Display File, with the most significant bit of the address set to '1'. Below this on the stack is the return address to the original calling routine.
- HL contains the address of the first character in the current line in the Display File, again with bit 15 set to '1'.
- The Z80's A register normally contains DD hex.

If , after decrementing C, there are still some scan lines to be generated for the current row of characters, the routine jumps to the POP DE instruction which clears the stack before jumping to LD R A. This instruction, together with the following Enable Interrupt instruction, ensures that 35 instructions later (35 is 100 hex - DD hex) a further interrupt will be caused as bit 6 of the refresh address becomes 0. This sets the basic scan line timing for the display.

Since HL is pointing to the first character of the line to be displayed, the JP (HL) instruction will make the Z80 fetch its next instruction from the Display File ! But, recognising that bit 15 of the address is a '1', the SCL chip will swing into action, storing the character code read from the Display File then forcing a NOP code onto the Z80 data lines D0-D7.

When it has realised that the instruction is a NOP, the Z80 will do nothing except fetch the next byte (from the Display File) as the next instruction to be executed. Again the SCL chip will

store the actual code read from the Display File, and give a NOP instruction to the Z80.

This process continues - the Z80 reading consecutive bytes out of the Display File and being fooled into thinking that they are NOPs - until a NEWLINE code is reached. The SCL chip, on seeing this code, stops feeding NOPs to the Z80, and allows it to see the NEWLINE byte. Since the NEWLINE code is 76 hex, the Z80's HALT instruction, the processor will stop doing anything except periodically generating refresh addresses until bit 6 of the refresh address reaches 0, causing an interrupt and so starting the next scan line.

After the last scan line of a particular row of characters, C becomes zero. The interrupt routine then pulls the address of the next byte in the Display File (the first character of the next row) from the stack into HL, and decrements the B register.

If B is now zero - indicating that there are no more rows left to display - then the interrupt routine will return to the calling routine. If B is not yet zero, however, then C is set back to 8 (by the SET 3 C instruction) and the display continues.

As each byte is read from the Display File it is stored in the SCL chip then, during the second (refresh) half of each NOP instruction cycle while the Z80 is putting the contents of its I and R registers onto the address lines, the SCL chip forces the low 9 address lines to the ROM to a pattern consisting of;

- The least 6 significant bits of the character code read from the Display File.
- Three bits defining which of the eight scan lines for each row of characters is being generated.

The high 7 address lines are the high 7 bits of the Z80's I register, which is set to 1E hex by the ZX81's power-up routines.

The ROM chip will then output the pattern of eight 'dots' which make up that scan line for that character. These are read into the SCL chip then fed out serially to the UHF modulator and so to your TV.

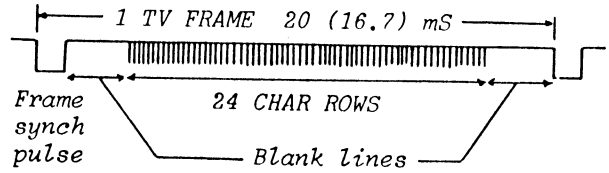
NMI Handler

As each horizontal TV scan line is completed in 64 micro-seconds, it takes 24 x 8 x 64 micro-seconds - which is just over 12 milli seconds - to output the complete 24 rows of characters displayed in each TV frame. But, to synchronise the TV set properly, each frame must last for 20 milli-seconds (16.7 milli-seconds for the U.S.A. model), so additional - blank - lines are needed to fill

in the top and bottom margins of the picture, before and after the frame synchronisation pulse;

VIDEO SIGNAL FOR ONE FRAME.

Line synchronisation pulses omitted for clarity.



When in the SLOW mode, the ZX81 uses the time occupied by these blank lines to carry on with your program. But, to keep track of the time, it is interrupted by the SCL chip every 64 micro-seconds. The Non-Maskable Interrupt is used for this function, and calls the routine starting at 0066 hex;

```
0066  08        NMI: EX AF AF
0067  3C             INC A          Increment A'
0068  FA 6D 00       JP M 006D       return to
006B  28 02          JR Z 006F        computation
006D  08             EX AF AF          if A' is
006E  C9             RET                not zero.
006F  08             EX AF AF
0070  F5             PUSH AF        Save
0071  C5             PUSH BC         working
0072  D5             PUSH DE          registers.
0073  E5             PUSH HL
0074  2A 0C 40       LD HL (400C)   Point HL to start
0077  CB FC          SET 7 H         of display file.
0079  76             HALT           Wait for int.
007A  D3 FD          OUT FD A       Turn SCL chip off.
007C  DD E9          JP (IX)        Return
```

The Z80 has two pairs of A and F registers, one pair is used for the normal functions of the ZX81, the other pair is reserved for use by the NMI routine.

Within the NMI routine, the A register is used to count the number of blank lines left to display - it is originally set to a negative number, and is incremented each line by the second instruction of the NMI routine. If A has not yet reached zero, the routine simply restores the original AF pair and returns to whatever the ZX81 was doing when it was interrupted.

When A does reach zero, the NMI routine saves the main registers on the stack, then loads HL with the address of the first byte of the display file - setting the most significant bit 15 to 1. The Z80 then halts until the next NMI interrupt is generated by

the SCL chip, when it returns to the instruction OUT FD A which turns off the SCL chip's NMI generation circuits. The Z80 then jumps to the return address held in the IX register.

Display Handler

Two subroutines, starting at 0292 and 02B5, are used to set up the ZX81 for the INT and NMI handlers.

```
Ø292  DD E1          POP IX            Put ret addr in IX
Ø294  FD 4E 28       LD C (IY+28)      C equals 'margin'
Ø297  FD CB 3B 7E    BIT 7 (IY+3B)     Jump if
Ø29B  28 ØC        ┌─JR Z Ø2A9          FAST mode.
Ø29D  79           │ LD A C            A'
Ø29E  ED 44        │ NEG                  equals
Ø2AØ  3C           │ INC A                  - 'margin'
Ø2A1  Ø8           │ EX AF AF
Ø2A2  D3 FE        │ OUT FE A          Enable SCL chip.
Ø2A4  E1           │ POP HL            Restore
Ø2A5  D1           │ POP DE             working
Ø2A6  C1           │ POP BC              registers
Ø2A7  F1           │ POP AF
Ø2A8  C9           │ RET               To computation.
Ø2A9  3E FC        └─LD A FC
Ø2AB  Ø6 Ø1          LD B 1
Ø2AD  CD B5 Ø2       CALL 02B5
Ø2BØ  2B             DEC HL
Ø2B1  E3             EX (SP) HL        Waste
Ø2B2  E3             EX (SP) HL         time.
Ø2B3  DD E9          JP (IX)           Return.

Ø2B5  ED 4F          LD R A            Set up for
Ø2B7  3E DD          LD A DD            FAST display
Ø2B9  FB             EI
Ø2BA  E9             JP (HL)           Start FAST display.
```

The second routine is called with HL pointing to a byte in the Display File, but with bit 15 set to '1', and with register B containing the number of rows of characters still to be displayed, and C containing the number of scan lines in the first row.

It simply loads A with the value DD - which will subsequently be loaded into the refresh register R of the Z80 at the start of each scan line - then enables the INT interrupt and starts the display with the JP (HL) instruction. The INT handler will then take over, generating the required display as described earlier, and on completion will return to the routine which called 02B5.

The first routine , from 0292 to 02B4, is more subtle and is called to generate the blank lines at the top and bottom of the display.

Having saved the return address to the calling routine in the IX register, and loaded the Z80's C register with the number of blank lines required, it then looks to see if it is in the Fast or the Slow mode.

If the ZX81 is in the Fast mode, the routine jumps to 02A9, where it uses the second routine at 02B5 to generate the blank lines (note that when called the HL register will be pointing to a NEWLINE code in the Display File).

However, if the ZX81 is in the Slow mode, then the instructions at 029D to 02A8 are executed. These load the A register with minus the number of blank lines needed, then enable the SCL chip's NMI generation circuit, then retrieve the Z80's main registers from the stack before returning to running your program - note that this routine starting at 0292 is itself called by the Main Display routine, which has previously stacked the Z80's main registers.

Having enabled the NMI circuits, the SCL chip will then cause an interrupt every 64 micro-seconds which as described earlier will just increment the second A register each time until it reaches zero. The INT handler will then return to the address contained in the IX register, which will be the return address for the 0292 routine, taking it back into the Main Display routine.

The overall effect of calling 0292 is therefore to generate a number of blank TV scan lines, and also to allow your program to proceed while they are being generated if the ZX81 is in the Slow mode. The number of blank lines generated is usually that contained in the System Variable MARGIN (4028).

Set up

This routine first looks at bits 6 & 7 of the System Variable CD-FLAG (403B) and returns if they are both the same.

If they are different, it then sees whether the machine being used is a ZX81 or the older ZX80 which does not have the Slow facility, by trying to enable the SCL chip's NMI gereration circuits with the OUT FE A instruction. It then looks to see if the A register had been affected. If not, then the computer must be a ZX80 (or a broken ZX81 !) and the routine jumps to 0226 to set bit 6 of CD-FLAG to zero before returning to the calling routine.

If the machine really is a ZX81, bit 7 of CD-FLAGS is set to '1' to indicate the Slow mode, then the main ZX80 registers are saved on the stack before jumping to the Main Display routine at 0229.

```
Ø2Ø7  21 3B 4Ø     LD HL 4Ø3B     CD flag
Ø2ØA  7E           LD A (HL)      Make carry 1 if
Ø2ØB  17           RLA             SLOW mode.
Ø2ØC  AE           XOR (HL)       Return if FAST
Ø2ØD  17           RLA             mode or ZX81 in
Ø2ØE  DØ           RET NC           SLOW mode.
Ø2ØF  3E 7F        LD A 7F
Ø211  Ø8           EX AF AF
Ø212  Ø6 11        LD B 11
Ø214  D3 FE        OUT FE A       Enable SCL chip.
Ø216  1Ø FE        DJNZ Ø216      Wait for int.
Ø218  D3 FD        OUT FD A       SCL chip off.
Ø21A  Ø8           EX AF AF
Ø21B  17           RLA
Ø21C  3Ø Ø8        JR NC Ø226     jump if not ZX81.
Ø21E  CB FE        SET 7 (HL)     Set SLOW mode.
Ø22Ø  F5           PUSH AF        Save
Ø221  C5           PUSH BC         working
Ø222  D5           PUSH DE          registers
Ø223  E5           PUSH HL
Ø224  18 Ø3        JR Ø229        To display routine.
Ø226  CB B6        RES 6 (HL)     Ret if machine is a
Ø228  C9           RET             ZX80, not a ZX81.
```

Main Display Loop

This routine is executed once for every TV frame displayed. It has several stages;

First, bits 0 to 14 of the System Variable FRAMES (4034/5) are decremented. If bit 15 is a 0 then FRAMES is also being used as the PAUSE counter, and control will be passed back to the calling routine by the RET NC instruction when bits 0 to 14 are all zeroes.

The ZX81 keyboard is then scanned by calling the subroutine at 02BB. This subroutine also starts the TV frame synchronisation pulse. The keyboard scan routine returns a value in the HL register pair corresponding to the key pressed, or FFFF if no key was pressed. This value is saved in the System Variable LAST-K (4025) then the following instructions see whether a valid keypress was detected. If so, and if the machine is in the Fast mode, the RET Z instruction at 0260 returns control to the calling routine. Otherwise, the instructions from 0264 to 0276 update the key debounce status System Variable at 4027.

The OUT FF A instruction at 0227 then ends the TV frame synchronisation pulse in time to start generating the next TV frame. The TV frame is generated in three parts, using the Display Handler routines described previously. First, the blank lines at the top of the screen are generated, then the 24 rows of char-

acters, then finally the blank lines at the bottom of the screen.

Having done this, the Z80 jumps back to the start of the Main Display routine at 0229 to begin the next round, and so it continues until either the end of a programmed PAUSE is reached, or - if in the Fast mode - until a valid keypress is detected. Of course, if you are in the Slow mode, the ZX81 is also executing your program in its 'spare' time during the blank lines at the top and bottom of the screen.

```
Ø229  2A 34 4Ø  ┌──→LD HL (4Ø34)     Decrement
Ø22C  2B        │   DEC HL           'FRAMES'
Ø22D  3E 7F     │   LD A 7F
Ø22F  A4        │   AND H
Ø23Ø  B5        │   OR L
Ø231  7C        │   LD A H
Ø232  2Ø Ø3     │  ┌JR NZ Ø237       Jmp if bits 0-14 not zero
Ø234  17        │  │RLA
Ø235  18 Ø2     │  ┌JR Ø239          C equals 0 if Pause.
Ø237  46        │  └→LD B (HL)
Ø238  37        │   SCF
Ø239  67        └──→LD H A
Ø23A  22 34 4Ø      LD (4Ø34) HL    Save new 'FRAMES'
Ø23D  DØ            RET NC          Ret if end of Pause.
Ø23E  CD BB Ø2      CALL Ø22B       Scan k/b & start frame synch.
Ø241  ED 4B 25 4Ø   LD BC (4Ø25)    Update
Ø245  22 25 4Ø      LD (4Ø25) HL     LAST-K
Ø248  78            LD A B          Set C
Ø249  C6 Ø2         ADD 2            if no key
Ø24B  ED 42         SBC HL BC         was pressed.
Ø24D  3A 27 4Ø      LD A (4Ø27)     Debounce status.
Ø25Ø  B4            OR H
Ø251  B5            OR L
Ø252  58            LD E B
Ø253  Ø6 ØB         LD B ØB
Ø255  21 3B 4Ø      LD HL 4Ø3B      CD-FLAG
Ø258  CB 86         RES Ø (HL)      Reset bit 0
Ø25A  2Ø Ø8        ┌JR NZ Ø264      Jmp if no keypress or if
Ø25C  CB 7E        │BIT 7 (HL)       key not same as before.
Ø25E  CB C6        │SET Ø (HL)
Ø26Ø  C8           │RET Z           Ret with valid keypress
Ø261  Ø5           │DEC B            if in Fast mode.
Ø262  ØØ           │NOP
Ø263  37           │SCF
Ø264  21 27 4Ø     └→LD HL 4Ø27      Debounce status.
Ø267  3F            CCF
Ø268  CB 1Ø         RL B
Ø26A  1Ø FE         DJNZ  Ø26A      Timing adjustment.
Ø26C  46            LD B (HL)
Ø26D  7B            LD A E
```

```
Ø26E  FE FE        CP FE
Ø27Ø  9F           SBC A A
Ø271  Ø6 1F        LD B 1F
Ø273  B6           OR (HL)
Ø274  AØ           AND B
Ø275  1F           RRA
Ø276  77           LD (HL) A
Ø277  D3 FF        OUT FF A        End frame synch
Ø279  2A ØC 4Ø     LD HL (4ØØC)    Point to Disp file.
Ø27C  CB FC        SET 7 H         Display
Ø27E  CD 92 Ø2     CALL Ø292        top margin.
Ø281  ED 5F        LD A R
Ø283  Ø1 Ø1 19     LD BC 19Ø1      Display
Ø286  3E F5        LD A F5          character
Ø288  CD B5 Ø2     CALL Ø2B5          rows.
Ø28B  2B           DEC HL          Display
Ø28C  CD 92 Ø2     CALL Ø292        bottom line.
Ø28F  C3 29 Ø2     JP Ø229         End of one TV frame.
```

Keyboard Scanning

This is a fairly straightforward subroutine which is normally called from the Main Display routine, but which can equally well be used by your machine language programs.

It returns a code in the HL register pair corresponding to the key pressed, or FFFF if no key was pressed. It also loads the System Variable MARGIN (4028) with the correct number of blank lines needed at the top and the bottom of the picture; 55 for U.K. machines, 31 for U.S.A. models, by detecting whether pin 22 of IC1 is strapped to 0V or not.

The routine also starts the frame synchronisation pulses, so an OUT FF A instruction must appear later to end it.

```
Ø2BB  21 FF FF     LD HL FFFF
Ø2BE  Ø1 FE FE     LD BC FEFE
Ø2C1  ED 78        IN A (C)        Look at keyboard
Ø2C3  F6 Ø1        OR Ø1
```

```
Ø2C5  F6 EØ      ┌→OR EØ
Ø2C7  57         │ LD D A
Ø2C8  2F         │ CPL            Make A reg equal 0
Ø2C9  FE Ø1      │ CP Ø1           if key pressed,
Ø2CB  9F         │ SBC A A          else FF
Ø2CC  BØ         │ OR B           S
Ø2CD  A5         │ AND L           C
Ø2CE  6F         │ LD L A           A
Ø2CF  7C         │ LD A H            N
Ø2DØ  A2         │ AND D              K
Ø2D1  67         │ LD H A              /
Ø2D2  CB ØØ      │ RLC B                B
Ø2D4  ED 78      │ IN A (C)       HL now FFFF if no keypress
Ø2D6  38 ED      └ JR C Ø2C5      Do it 7 times.
Ø2D8  1F           RRA
Ø2D9  CB 14        RL H           Calculate
Ø2DB  17           RL A            the
Ø2DC  17           RL A             margin
Ø2DD  17           RL A              (55
Ø2DE  9F           SBC A A            for
Ø2DF  E6 18        AND 18              U.K.)
Ø2E1  C6 1F        ADD 1F
Ø2E3  32 28 4Ø     LD (4Ø28) A    Save it.
Ø2E6  C9           RET
```

The hexadecimal values returned in the HL register for the 39 keys are as shown in the table below. 0100 hex should be subtracted from the values given when the SHIFT key is used.

KEY	VALUE	KEY	VALUE	KEY	VALUE
1	FDF7	R	EFFB	J	EFBF
2	FBF7	T	DFFB	K	F7BF
3	F7F7	Y	DFDF	L	FBBF
4	EFF7	U	EFDF	N/L	FDBF
5	DFF7	I	F7DF	Z	FBFE
6	DFEF	O	FBDF	X	F7FE
7	EFEF	P	FDDF	C	EFFE
8	F7EF	A	FDFD	V	DFFE
9	FBEF	S	FBFD	B	DF7F
Ø	FDEF	D	F7FD	N	EF7F
Q	FDBF	F	EFFD	M	F77F
W	FBFB	G	DFFD	.	FB7F
E	F7FB	H	DFBF	space	FD7F

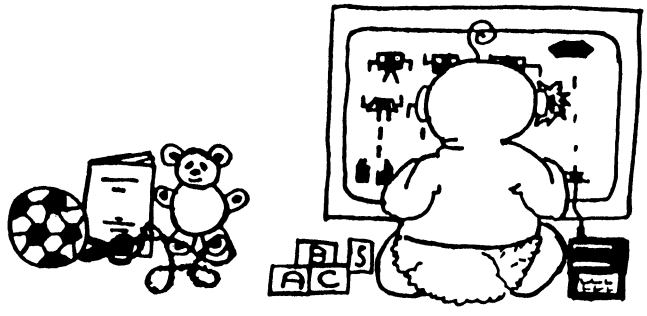
A second subroutine, at 07BD, translates the 16 bit value into a key number 0-78. It should be called with the 16 bit value in the BC register pair, and returns with the key number in the A

register, and HL pointing to the appropriate byte of the key table (007E to 00CB), and the carry flag C set to 1 if only one key had been pressed.

```
Ø7BD  16 ØØ          LD D ØØ
Ø7BF  CB 28          SRA B        Make
Ø7C1  9F             SBC A A       A
Ø7C2  F6 26          OR 26          equal
Ø7C4  2E Ø5          LD L Ø5         21 if shift
Ø7C6  95             SUB L            else FA
Ø7C7  85          ┌─►ADD A L     A now equals
Ø7C8  37          │  SCF          21 of FF
Ø7C9  CB 19       │  RRC           +5+5+5 ....
Ø7CB  38 FA       └─ JR C
Ø7CD  ØC             INC C        Return with carry 0
Ø7CE  CØ             RET NZ        if not 1 key pressed
Ø7CF  48             LD C B
Ø7DØ  2D             DEC L
Ø7D1  2E Ø1          LD L Ø1
Ø7D3  2Ø F2       ┌─ JR NZ
Ø7D5  21 7D ØØ    │  LD HL ØØ7D
Ø7D8  5F          │  LD E A
Ø7D9  19          │  ADD HL DE
Ø7DA  37          │  SCF
Ø7DB  C9          │  RET         With carry set.
```

Chapter 4

Some Games, and Other Novelties



Weekday

This program calculates the day of the week for any date after 1750 AD (except for Russia, which did not change to the Gregorian calendar until 1917).

Monday's child is fair of face,
Tuesday's child is full of grace.
Wednesday's child is full of woe,
Thursday's child has far to go.
Friday's child is loving and giving,
Saturday's child works hard for a living.
But the child that's born on the Sabbath day
Is bonny, blithe, and good, and gay.

```
10 DIM K(12)
20 LET A$="033614625035"
30 FOR A=1 TO 12
40 LET K(A)=CODE A$(A)-CODE "0"
50 NEXT A
60 LET A$="SUNDAY   MONDAY   TUESDAY  WEDNESDAYTHURSDAY
   FRIDAY   SATURDAY "

100 CLS
110 PRINT "YEAR ?"
120 INPUT Y
130 PRINT AT 0,5;Y,,"MONTH (1-12) ?"
140 INPUT M
150 PRINT AT 1,6;M;"       ",,"DAY ?"
160 INPUT D
170 PRINT AT 2,4;D

200 LET B=Y-100*INT (Y/100)
210 LET X=INT (Y/100)-15
220 LET A=3+X*21
230 LET I=INT ((A-28*INT (A/28))/4)
240 LET A=B*5+4*(D+K(M)+I)
250 LET W=1+INT ((A-28*INT (A/28))/4)
260 IF M>2 THEN GOTO 320
270 LET L=X-1+4*INT ((X-1)/4)
280 IF B>0 THEN LET L=Y-4*INT(Y/4)
290 IF L>0 THEN GOTO 320
300 IF W=0 THEN LET W=6
310 LET W=W-1
320 IF W=0 THEN LET W=7
330 PRINT AT 4,0;"IS A ";A$(9*W-8 TO 9*W)
```

There should be exactly 63 characters including spaces between the quotation marks in line 60.

Dynamite

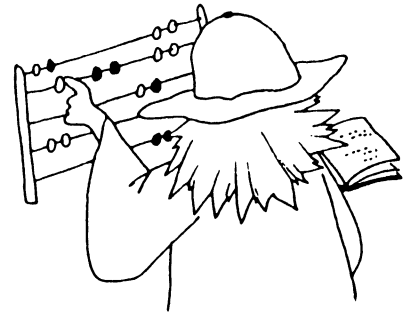
The game starts with a row of very old, very dangerous, sticks of dynamite. You and the ZX81 take turns to remove them, but only 1,2 or 3 sticks may be taken at a time. The last stick will explode, so the player left with this one loses !

```
 1Ø REM DYNAMITE
 2Ø SLOW
1ØØ GOSUB 2ØØØ
11Ø RAND
12Ø LET D=21+INT (12*RND)
13Ø LET S=Ø
14Ø FOR A=Ø TO D-1
15Ø FOR B=9 TO 6 STEP -1
16Ø PRINT AT B,A;"■"
17Ø NEXT B
18Ø NEXT A
19Ø PRINT AT 14,Ø;D;" STICKS OF DYNAMITE"
2ØØ PRINT AT 16,Ø;"HOW MANY STICKS WILL YOU TAKE ?"
21Ø LET T=CODE INKEY$-28
22Ø IF T<1 OR T>3 OR T>D THEN GOTO 21Ø
23Ø PRINT AT 16,Ø;S$;AT 16,Ø;"YOU TAKE ";T
24Ø LET D=D-T
25Ø FOR A=S+D+T-1 TO S+D STEP -1
26Ø FOR B=6 TO 9
27Ø PRINT AT B,A;" "
28Ø NEXT B
29Ø NEXT A
3ØØ PRINT AT 14,Ø;D;" ";AT 14,22;"LEFT"
31Ø IF D=Ø THEN GOTO 7ØØ
32Ø IF D=1 THEN GOTO 8ØØ
33Ø PRINT AT 16,Ø;"PRESS ANY KEY FOR MY TURN";S$
34Ø IF INKEY$="" THEN GOTO 33Ø
35Ø LET R=D-4*INT (D/4)
36Ø LET C=R+3-4*INT ((R+3)/4)
37Ø IF R=1 THEN LET C=1+INT (3*RND)
38Ø LET D=D-C
39Ø PRINT AT 16,Ø;S$;AT 16,Ø;"I TAKE ";C
4ØØ FOR A=S TO S+C-1
41Ø FOR B=6 TO 9
42Ø PRINT AT B,A;" "
43Ø NEXT B
```

```
44Ø NEXT A
45Ø PRINT AT 14,Ø;D;" "
46Ø LET S=S+C
47Ø IF D=Ø THEN GOTO 8ØØ
48Ø IF D=1 THEN GOTO 7ØØ
49Ø GOTO 2ØØ

7ØØ GOSUB 1ØØØ
71Ø PRINT AT 16,Ø;"*** I WIN ***";AT 18,Ø;X$(INT (1+5*RND))
72Ø GOTO 9ØØ

8ØØ GOSUB 1ØØØ
81Ø PRINT AT 16,Ø;"** YOU HAVE WON **";AT 18,Ø;Y$(INT (1+5*RND))

9ØØ PRINT AT 2Ø,Ø;"ANOTHER GAME ?"
91Ø IF INKEY$="N" THEN STOP
92Ø IF INKEY$<>"Y" THEN GOTO 91Ø
93Ø CLS
94Ø RUN

1ØØØ LET A=6
1Ø1Ø PRINT AT 14,Ø;S$;AT 16,Ø;S$;AT 7,S;" ";AT 8,S;" ";AT 9,S;" "
1Ø2Ø FOR B=Ø TO A
1Ø3Ø PRINT AT A-B,S-B;"*";AT A+B,S-B;"*";AT A+B,S+B;"*";AT A-B,
     S+B;"*"
1Ø4Ø NEXT B
1Ø5Ø FOR B=Ø TO A
1Ø6Ø PRINT AT A-B,S-B;" ";AT A+B,S-B;" ";AT A+B,S+B;" ":AT A-B,
     S+B;" "
1Ø7Ø NEXT B
1Ø8Ø RETURN

2ØØØ DIM X$(5,32)
2Ø1Ø DIM Y$(5,32)
2Ø2Ø DIM S$(32)
2Ø3Ø LET X$(1)="WHAT A CLEVER COMPUTER I AM"
2Ø4Ø LET X$(2)="**** ZX RULES, OK ? ****"
2Ø5Ø LET X$(3)="HOW ABOUT THAT THEN"
2Ø6Ø LET X$(4)="- OF COURSE."
2Ø7Ø LET X$(5)="WHY DONT YOU GIVE UP ?"
2Ø8Ø LET Y$(1)="** CONGRATULATIONS **"
2Ø9Ø LET Y$(2)="I BET YOU CHEATED"
21ØØ LET Y$(3)="I WILL GET YOU NEXT TIME"
211Ø LET Y$(4)="OF COURSE I LET YOU"
212Ø LET Y$(5)="WHAT LUCK"
213Ø RETURN
```

The actual calculations involved in this program are relatively simple, most of the program being devoted to giving an interesting display as the game progresses. Note how lines 710,810 give a variable response at the end of the game; the comments in lines 2000-2130 may of course be changed to suit the temperament of your particular ZX81.

Sums Tester 1K



And now to prove that the ZX81 has an educational value; SUMS TESTER provides exercises in simple addition.

It poses 10 questions of the form A + B 1 ?, and displays a running total of the number of correct answers. At the end of the ten questions it shows the total time taken to answer them all.

```
 1Ø RAND
 2Ø LET S=Ø
 3Ø LET T=S
1ØØ FOR G=1 TO 1Ø
11Ø PRINT ,,"PRESS Y FOR QUESTION ";G
12Ø IF INKEY$<>"Y" THEN GOTO 12Ø
13Ø CLS
14Ø LET A=INT (99*RND+1)
15Ø LET B=INT (99*RND+1)
16Ø POKE 16436,255
17Ø POKE 16437,255
18Ø PRINT ,,A;" + ";B;" = ?"
19Ø INPUT C
2ØØ LET T=T+65535-PEEK 16436-256*PEEK 16437
21Ø IF C=A+B THEN GOTO 24Ø
22Ø PRINT ,,C;" IS WRONG, THE ANSWER IS ";A+B
23Ø GOTO 3ØØ
24Ø PRINT ,,C;" IS RIGHT"
25Ø LET S=S+1
3ØØ PRINT ,,"SCORE = ";S
31Ø NEXT G
32Ø PRINT ,,"YOU TOOK ";INT (T/5Ø);" SECONDS"
```

U.S. readers should change '50' in line 320 to '60'.

As given the program only asks questions of addition, with numbers from 1 to 99. Why not modify it for subtraction, multiplication, or division, or to use a smaller or larger range of numbers ?

ZX Sofft Shoppes

How do you rate your chances as a Super Software Salesman ? This program gives you the chance to try running your own business. It can be played by one individual, or - better - by several competing players.

The program gives you successive months in the life of a typical software shop. At the start of each month you have an amount of cash, and unsold stock left over from the previous months. You have to decide how much to spend on advertising, how much on new stock, and what mark-up to apply (Mark-up is selling price divided by cost, thus if you buy some software for 1.00 and sell it for 1.50 your 'mark-up' is 1.5).

At the end of each 'month', the ZX Sofft Shoppe computer shows you the sales for the month - in terms of both stock value (what you paid for it) and selling price.It then displays the value of the remaining stock and your latest cash position.

Tax is payable in January on the previous year's profits. If you can't pay your tax bill then you are declared bankrupt and out of the game. As an aid to budgetting, the current tax liability for the year to date is included in each month's results; if no tax liability is shown then you are making a loss !

The business model used by the program gives higher sales around Christmas, and lower than average sales in the summer months. Both advertising expenditure and mark-up will affect your sales, as will competition from other shops. To maximise each month's sales you should keep a balance between old and new stock.

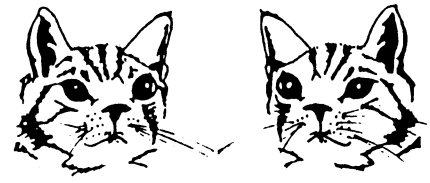
Have fun. Anyone tripling his initial investment in the first year is invited to apply to Timedata for an interesting job !

```
 100 PRINT AT 5,4;"YE ZX SOFFT SHOPPES"
 110 GOSUB 5000
1000 FOR X=1 TO N
1010 CLS
1020 PRINT N$(X)
1030 IF M=0 THEN GOTO 1600
1100 PRINT AT 2,0;"RESULTS FOR ";M$(M);AT 2,23;Y
1110 LET F=A(X,1)+A(X,2)+A(X,3)+A(X,5)
1120 LET B=INT ((3+RND)*A*A(X,5)/(A(X,4)**3))
1130 LET C=INT (B*(1+RND)*.5)
1140 IF B>A(X,3) THEN LET B=A(X,3)
1150 IF C>A(X,2)THEN LET C=A(X,2)
1160 LET A(X,2)=A(X,2)-C+A(X,3)-B
1170 LET A(X,3)=0
1180 LET D=INT ((B+C)*A(X,4))
1190 LET A(X,1)=A(X,1)+D
```

```
12ØØ LET T=(A(X,1)+A(X,2)-F)*.3
121Ø PRINT AT 4,Ø;B+C;"  STOCK SOLD FOR";TAB 24;D
122Ø IF M>1 OR Y=1982 THEN GOTO 154Ø
123Ø IF A(X,6)<=A(X,1) THEN GOTO 15ØØ
13ØØ PRINT AT 5,Ø;"TAX DUE";TAB 24;A(X,6)
131Ø PRINT AT 7,Ø;"BUT YOU HAVE ONLY";TAB 24;A(X,1)
132Ø PRINT AT 9,Ø;"SO YOU ARE BANKRUPT,"
133Ø PRINT "AND OUT OF THE GAME."
134Ø IF N=1 THEN STOP
14ØØ IF X=N THEN GOTO 147Ø
141Ø FOR B=X TO N-1
142Ø FOR C=1 TO 6
143Ø LET A(B,C)=A(B+1,C)
144Ø NEXT C
145Ø LET N$(B)=N$(B+1)
146Ø NEXT B
147Ø LET N=N-1
148Ø LET X=X-1
149Ø GOTO 1999
15ØØ IF A(X,6)<Ø THEN LET A(X,6)=Ø
151Ø PRINT AT 5,Ø;"TAX PAID;";TAB 24;INT A(X,6)
152Ø LET A(X,1)=A(X,1)-INT A(X,6)
153Ø LET A(X,6)=Ø
154Ø LET A(X,6)=A(X,6)+T
16ØØ PRINT AT 7,Ø;"YOU NOW HAVE;";TAB 18;"CASH";TAB 24;A(X,1);
     TAB 18;"STOCK";TAB 24;A(X,2)
161Ø IF A(X,6)>Ø THEN PRINT AT 1Ø,Ø;"TAX DUE NEXT JANUARY;";
     TAB 24;INT A(X,6)
162Ø LET U=M+1
163Ø LET V=Y
164Ø IF U<13 THEN GOTO 167Ø
165Ø LET U=1
166Ø LET V=V+1
167Ø PRINT AT 13,Ø;"DECISIONS FOR ";M$(U);AT 13,23;V
17ØØ PRINT AT 15,Ø;"HOW MUCH WILL YOU SPEND;";TAB 9;"ON ADVE
     RTISING ?"
171Ø INPUT I
172Ø LET I=INT I
1730 IF I<Ø OR I>A(X,1) THEN GOTO 171Ø
174Ø LET A(X,5)=I
175Ø LET A(X,1)=A(X,1)-I
176Ø PRINT AT 16,24;I
177Ø PRINT AT 17,15;"ON STOCK ?"
178Ø INPUT I
179Ø LET I=INT I
18ØØ IF I<Ø OR I>A(X,1) THEN GOTO 178Ø
181Ø LET A(X,3)=I
182Ø LET A(X,1)=A(X,1)-I
183Ø PRINT AT 17,24;I
```

```
184Ø PRINT "MARKUP (PRICE/COST)     ?"
185Ø INPUT I
186Ø IF I<Ø THEN GOTO 185Ø
187Ø LET A(X,4)=I
188Ø PRINT AT 18,24;I
19ØØ PRINT AT 21,Ø;"DO YOU WANT TO CHANGE ANYTHING ?"
191Ø INPUT I$
192Ø IF (I$+Z$)(1)="N" THEN GOTO 1999
193Ø FOR I=15 TO 21
194Ø PRINT AT I,Ø;Z$
195Ø NEXT I
196Ø LET A(X,1)=A(X,1)+A(X,5)+A(X,3)
197Ø GOTO 17ØØ
1999 NEXT X
2ØØØ LET A=Ø
2Ø1Ø FOR X=1 TO N
2Ø2Ø LET A=A+A(X,4)
2Ø3Ø NEXT X
2Ø4Ø LET A=SQR (A/N)
21ØØ LET M=M+1
211Ø IF M<13 THEN GOTO 1ØØØ
212Ø LET M=1
213Ø LET Y=Y+1
214Ø GOTO 1ØØØ
5ØØØ REM INITIALISATION
5Ø1Ø DIM M$(12,11)
5Ø2Ø LET M$(1)="JANUARY"
5Ø3Ø LET M$(2)="FEBRUARY"
5Ø4Ø LET M$(3)="MARCH"
5Ø5Ø LET M$(4)="APRIL"
5Ø6Ø LET M$(5)="MAY"
5Ø7Ø LET M$(6)="JUNE"
5Ø8Ø LET M$(7)="JULY"
5Ø9Ø LET M$(8)="AUGUST"
51ØØ LET M$(9)="SEPTEMBER"
511Ø LET M$(1Ø)="OCTOBER"
512Ø LET M$(11)="NOVEMBER"
513Ø LET M$(12)="DECEMBER"
514Ø LET Y=1982
515Ø LET M=Ø
516Ø DIM Z$(32)
52ØØ PRINT AT 15,Ø;"NUMBER OF SHOPS (PLAYERS) ?"
521Ø INPUT N
522Ø DIM N$(N,32)
523Ø DIM A(N,6)
524Ø FOR X=1 TO N
525Ø PRINT AT 15,Ø;Z$;AT 15,Ø;"NAME OF SHOP ";X
526Ø INPUT N$(X)
527Ø LET A(X,1)=1ØØ
528Ø NEXT X
529Ø RETURN
```

Copycat 1K



In this version of the well known childrens' game, a sequence of letters is flashed onto the TV screen. You are then invited to key in the same sequence. This happens ten times, each time with a longer sequence, and your score of correct responses is shown. When the game starts you can choose the speed at which the letters are shown; from 1 (boring) to 5 (impossible ?). After each go, press key 'Y' for the next sequence, or BREAK to quit.

```
 10 SLOW
 20 LET C=0
 30 PRINT "SPEED (1-5)"
 40 INPUT S
 50 LET S=55-(S AND S>=1 AND S<=5)*10

100 FOR G=1 TO 10
110 CLS
120 LET A$=""
130 FOR B=1 TO G+1
140 LET C$=CHR$ (38+25*RND)
150 PRINT AT 1,1;C$
160 LET A$=A$+C$
170 FOR X=1 TO S
180 NEXT X
190 PRINT AT 1,1;" "

200 FOR X=1 TO 10
210 NEXT X
220 NEXT B

300 PRINT "WHAT WERE THEY ?"
310 INPUT C$
320 PRINT C$
330 IF C$<>A$ THEN GOTO 370
340 PRINT "CORRECT"
350 LET C=C+1
360 GOTO 390
370 PRINT "WRONG, THEY WERE;"
380 PRINT A$
390 PRINT AT 7,1;C;" OUT OF ";G;(" NEXT GO ?" AND G<10)

400 PRINT "WELL DONE" AND C=10
410 IF G=10 THEN STOP
420 IF INKEY$<>"Y" THEN GOTO 420
430 NEXT G
```

Note that FOR-NEXT loops are used to generate delays in the program (lines 170,180,200,210) rather than the ZX81 PAUSE command, which can give an annoying flicker in this type of application.

BINDECHEX 1K

This program will accept a binary, hexadecimal, or decimal number and display it in all three forms.

Binary numbers can be up to 16 digits long, and should be entered starting with the letter 'B' ;

B1ØØ111ØØ

Hexadecimal numbers can be up to 4 digits long and should be entered starting with 'H';

HFF1A

Decimal numbers can be from Ø to 65535, and should start with the letter 'D';

D4Ø96

The program will keep asking for more numbers to convert until you enter a null line (just press NEWLINE/ENTER).

```
 1Ø LET Z=PI-PI
 2Ø LET U=PI/PI
 3Ø LET T=U+U
 4Ø LET F=T+T
 5Ø LET S=F*F
1ØØ PRINT AT F,Z;"ENTER NUMBER PREFIXED";TAB F;"B IF BINARY"
    ;TAB F;"D IF DECIMAL";TAB F;"H IF HEX"
11Ø INPUT I$
12Ø IF I$="" THEN STOP
13Ø LET T$=I$(U)
14Ø LET X=(T AND T$="B")+(1Ø AND T$="D")+(S AND T$="H")
15Ø IF X=Z THEN GOTO 11Ø
16Ø CLS
2ØØ LET D=Z
21Ø FOR A=T TO LEN I$
22Ø LET D=D*X+CODE I$(A)-28
23Ø NEXT A
3ØØ PRINT " DEC   HEX      BINARY";AT T,Z;D
31Ø LET E=D
32Ø FOR A=U TO F
33Ø PRINT AT T,1Ø-A;CHR$ (CODE "Ø"+E-S*INT (E/S))
34Ø LET E=INT (E/S)
35Ø NEXT A
36Ø FOR A=U TO S
37Ø PRINT AT T,27-A;CHR$ (CODE "Ø"+D-T*INT (D/T))
38Ø LET D=INT (D/T)
39Ø NEXT A
4ØØ GOTO 1ØØ
```

The program first converts whatever form of number is entered to the decimal equivalent D, then prints that and the hex and binary versions - calculating them as it goes (lines 320-390).

It just squeezes into 1K ; note lines 10-50 which remove the need for RAM consuming constants Ø,1,2,4 and 16 throughout the program.

Why not try modifying the program to work in other bases, say Octal (base 8) or Ternary (base 3) ?

Roman–Arabic 1K

This program accepts a number written in Roman numerals (for example MCMLXXXII) and converts it to our modern Arabic numeral form. Having converted one number, it will ask for another ; enter a null line (just press NEWLINE/ENTER) to end.

Note that the Roman numerals are the letters M,D,C,L,X,V and I. The program will not accept the Arabic numeral '1'.

```
 1Ø PRINT AT 5,Ø;"ENTER ROMAN FORM OF NUMBER"
 2Ø LET P=Ø
 3Ø INPUT D$
 4Ø IF D$="" THEN STOP
 5Ø CLS
 6Ø LET Y=P
 7Ø FOR A=LEN D$ TO 1 STEP-1
 8Ø FOR B=1 TO 7
 9Ø IF D$(A)="IVXLCDM"(B) THEN GOTO 13Ø
1ØØ NEXT B
11Ø PRINT D$;" NOT VALID"
12Ø GOTO 1Ø
13Ø LET V=VAL "ØØ1ØØ5Ø1ØØ5Ø1ØØ5ØØ1E3"(B*3-2 TO B*3)
14Ø LET Y=Y+(V AND B>=P)-(V AND B<P)
15Ø IF B>P THEN LET P=B
16Ø NEXT A
17Ø PRINT D$;" = ";Y
18Ø GOTO 1Ø
```

Note how line 90 provides a fixed set of characters, and line 130 similarly provides a fixed set of numbers.

Arabic–Roman 1K

This program converts our customary arabic numbers to their Roman equivalents, e.g;

1982 to MCMLXXXII

After converting one number, it asks for another. Enter STOP to quit.

```
 2 PRINT AT 5,Ø;"ENTER NUMBER"
 4 INPUT N
 6 CLS
 8 PRINT N;" = ";
1Ø FOR A=1 TO 7
12 LET B=VAL "1E35ØØ1ØØØ5ØØ1ØØØ5ØØ1"(3*A-2 TO 3*A)
14 IF N<B THEN GOTO 22
16 PRINT "MDCLXVI"(A);
18 LET N=N-B
2Ø GOTO 12
22 LET C=VAL "1ØØ1ØØØ1ØØ1ØØØ1ØØ1ØØØ"(3*A-2 TO 3*A)
24 IF N<B-C THEN GOTO 32
26 PRINT "CCXXIII"(A);
28 LET N=N+C
3Ø GOTO 12
32 NEXT A
34 GOTO 2
```

Note how lines 12,16,22 and 26 provide fixed sets of numbers.

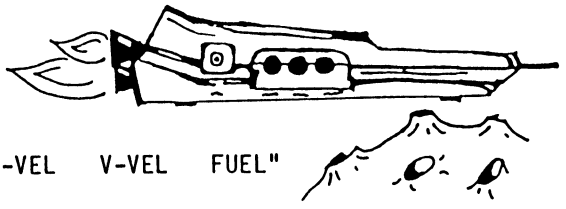
Fencing 1K

This program draws a sequence of basket-weave patterns. Use the BREAK key to stop it.

```
 1Ø SLOW
 2Ø LET H=1
 3Ø LET V=1
 4Ø LET X=54*RND
 5Ø LET Y=Ø
1ØØ FOR I=1 TO 5ØØ
11Ø PLOT X,Y
12Ø LET H=H-(2 AND X>54)+(2 AND X<1)
13Ø LET V=V-(2 AND Y>3Ø)+(2 AND Y<1)
14Ø LET X=X+H
15Ø LET Y=Y+V
16Ø NEXT I
2ØØ CLS
21Ø RUN
```

Moonlander

You are piloting a rocket ship about to land on the Moon (the @@@*** auto-landing computer has failed again !). At the start of the game you are at a height of 370 metres, travelling horizontally at 30 metres per second, and just starting to come out of orbit and fall towards the Moon's surface. You can control the ship by pressing keys 5 and 7 (one at a time) to apply left or vertical thrust respectively. Try to land as near as possible to the white square at the bottom of the screen with a vertical velocity of less than 3 metres per second. And watch your fuel !



```
100 LET A=370
110 LET H=30
120 LET V=0
130 LET F=55
140 LET P=0

200 PRINT "HEIGHT  H-VEL   V-VEL   FUEL"
210 FOR I=0 TO 63
220 PLOT I,0
230 NEXT I
240 UNPLOT 32,0

300 FOR I=1 TO 1E9
310 PLOT P,A/10
320 PRINT AT 1,0;A;" ";TAB 8;H;" ";TAB 16;V;" ";TAB 24;F;" "
330 IF A<1 OR P<0 OR P>63 THEN GOTO 1000
340 LET A$=INKEY$
350 IF F<=0 THEN GOTO 400
360 LET V=V+1-(2 AND A$="7")
370 LET H=H-(A$="5")
380 LET F=F-(A$="7" OR A$="5")

400 LET A=A-V
410 LET P=P+H/25
420 NEXT I

1000 IF ABS V<3 THEN PRINT AT 10,0;"A SAFE LANDING"
1010 IF ABS V>=3 AND ABS V<5 THEN PRINT AT 10,0;"BUMPY, BUT WE
     ARE STILL ALIVE"
1020 IF ABS V>=5 THEN GOTO 1100
1030 IF ABS (P-32)>=3 THEN PRINT "A LONG WAY FROM BASE"
1040 IF ABS (P-32)>=6 THEN PRINT "WHICH COUNTRY IS THIS THEN ?"
1050 IF ABS V<3 AND ABS (P-32)<3 THEN PRINT "HELLO BUCK ROGERS"
1060 STOP
1100 PRINT AT 10,0;"ANOTHER CRATER"
```

Note how lines 340,360 and 370 vary the vertical velocity V, the horizontal velocity H and the fuel remaining F according to whether keys 5 or 7 are pressed. You could expand this section of the program to give right thrust by detecting key 8.

Things

?? =

You think of the name of a 'thing', the computer then tries to find out what it is by asking a series of questions.

If the computer doesn't know what you are thinking of, it will eventually give up and ask you what it was, and for a question that it can ask in future to distinguish this new 'thing' from the computer's guess (each question must have a simple Yes/No answer; "Does it have 4 legs" is a valid type of question but "How many legs has it" is not). In this way the computer can build up it's store of 'knowledge' as you play by adding to a data file, which can be saved on tape for future sessions.

This data file can be constructed around virtually any subject by a suitable choice of questions and answers; why not try to construct one for simple automobile fault diagnosis, or whatever your imagination suggests. It is saved on tape as part of the program - as usual with ZX81 data files - and you can choose the name it is filed under at the time of saving it, so different data files can be generated by the program. To use the file later, just LOAD as normal. If you want to delete the current data file from the program in RAM, just stop it with the BREAK key, then RUN to start it again but with an empty data bank.

```
 10 REM THINGS
 20 LET A=9999
 30 DIM A$(A)
 40 LET A$(1 TO 20)="Q09002100037IS IT BIG"
 50 LET A$(21 TO 36)="T13SPACE SHUTTLE"
 60 LET A$(37 TO 43)="T04ZX81"
 70 LET F=44

100 CLS
110 LET P=1
120 PRINT AT 8,8;"THINK OF A THING.";AT 11,13;"READY ?"
130 IF INKEY$<>"Y" THEN GOTO 130

200 CLS
210 LET L=VAL (A$(P+1 TO P+2))
```

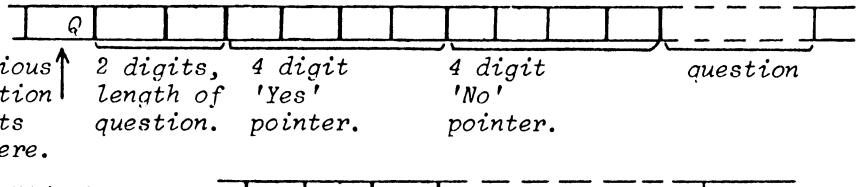
```
22Ø IF A$(P)="T" THEN GOTO 3ØØ
23Ø PRINT AT 8,Ø;A$(P+11 TO P+1Ø+L);" ?"
24Ø GOSUB 1ØØØ
25Ø LET Q=P+3+(4 AND I$="N")
26Ø LET P=VAL A$(Q TO Q+3)
27Ø GOTO 2ØØ

3ØØ PRINT AT 8,Ø;"IS IT A";AT 1Ø,Ø;A$(P+3 TO P+2+L);" ?"
31Ø GOSUB 1ØØØ
32Ø IF I$="N" THEN GOTO 4ØØ
33Ø PRINT AT 12,Ø;"I GUESSED IT, WHAT A CLEVER ZX81"
34Ø GOTO 8ØØ

4ØØ PRINT AT 13,Ø;"WHAT WAS IT THEN ?"
41Ø INPUT T$
42Ø CLS
43Ø PRINT AT 6,Ø;"WHAT QUESTION DISTINGUISHES A"
44Ø PRINT AT 8,Ø;A$(P+3 TO P+2+L);TAB 1;TAB Ø;"FROM A";
    TAB 1;TAB Ø;T$;" ?"
45Ø INPUT Q$
46Ø PRINT AT 14,Ø;Q$
47Ø IF F+LEN T$+LEN Q$>A THEN GOTO 95Ø
48Ø PRINT

5ØØ PRINT "IS THE ANSWER Y OR N FOR A";TAB Ø;T$;" ?"
51Ø GOSUB 1ØØØ
515 CLS
52Ø LET X=F
53Ø GOSUB 11ØØ
54Ø LET A$(Q TO Q+3)=X$
55Ø LET X=LEN Q$
56Ø GOSUB 11ØØ
57Ø LET A$(F TO F+1Ø+X)="Q"+X$(3 TO)+"12345678"+Q$

6ØØ LET X=P
61Ø GOSUB 11ØØ
62Ø LET P$=X$
63Ø LET Q=F
64Ø LET F=F+LEN Q$+11
65Ø LET X=F
66Ø GOSUB 11ØØ
67Ø LET X$=X$+P$
68Ø IF I$="N" THEN LET X$=X$(5 TO )+X$( TO 4)
69Ø LET A$(Q+3 TO Q+1Ø)=X$

7ØØ LET X=LEN T$
71Ø GOSUB 11ØØ
72Ø LET A$(F TO F+X+2)="T"+X$(3 TO )+T$
73Ø LET F=F+X+3

8ØØ PRINT AT 2Ø,Ø;"ANOTHER GO ?"
81Ø GOSUB 1ØØØ
82Ø IF I$="Y" THEN GOTO 1ØØ
830 CLS
840 PRINT AT 2Ø,Ø;"DO YOU WANT TO SAVE THIS DATA ?"
```

```
85Ø GOSUB 1ØØØ
86Ø CLS
87Ø IF I$="N" THEN STOP
88Ø PRINT AT 18,Ø;"TAPE FILE NAME ?"
89Ø INPUT T$
9ØØ PRINT AT 2Ø,Ø;"START RECORDER,THEN PRESS Y"
91Ø GOSUB 1ØØØ
92Ø SAVE T$
93Ø GOTO 1ØØ
95Ø PRINT AT 19,Ø;"MY MEMORY IS FULL"
96Ø GOTO 8ØØ
1ØØØ LET I$=INKEY$
1Ø1Ø IF I$<>"Y" AND I$<>"N" THEN GOTO 1ØØØ
1Ø2Ø RETURN
11ØØ LET X$="ØØØØ"+STR$ X
111Ø LET X$=X$(LEN X$-3 TO )
112Ø RETURN
```

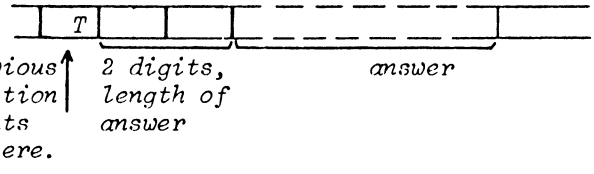
This program illustrates the use of a 'tree' structure for data storage. Each question stored has two associated pointers which give the locations of the next logical questions for 'Y' and 'N' responses. The program simply moves from one question to the next according to each 'Y' or 'N' response until it reaches a possible answer. If this answer is not correct, the correct answer is obtained from the user, together with a new question. These are stored in the first available free memory space, then the pointers are altered to reflect these additions.

The questions and answers are actually stored in a single dimensioned character array A$(); new items being added onto the end of the previous data, in 'linked list' form. This is an efficient way of storing a lot of variable length data, although it may not be the best solution for programs which need to access individual data elements quickly as to find any particular item the program has to search through the list from the beginning.

Questions are stored as;



and answers as;



See lines 40,50 & 60 for examples.

Buzz 1K

An aid for the busy executive, BUZZ generates impressive sounding phrases for use in reports to higher management. Or it could be used for writing newspaper headlines, or for choosing names for the characters in TV plays, depending on the data entered.

This data consists of 30 words or phrases (not more than 10 characters each), numbered 1 to 30. They are divided into three groups, numbers 1 to 10, 11 to 20 and 21 to 30. The program prints one word or phrase from each group in turn, selecting them at random.

When the program is first RUN, you should enter the 30 words or phrases in turn, the number of the phrase to be entered is displayed as a prompt each time.

After all 30 words or phrases have been entered, the program will display the first of it's compositions. It will generate another one if you press the "Y" key, pressing any other key will stop the program, giving report 9/210 . CONT will then save the program (complete with the data you have entered) to tape with the name "BUZZ" - change line 220 if you prefer another name - so that when it is subsequently LOADed from tape the program will automatically run using the data that has been entered previously.

A suitable list of words is;

1 GLOBAL	11 SILICON	21 TECHNOLOGY
2 ECOLOGICAL	12 TOP DOWN	22 PROJECTION
3 PARALLEL	13 PRAGMATIC	23 PHASE
4 VALID	14 MODULAR	24 SCENARIO
5 FEASIBLE	15 REDUNDANT	25 PROJECT
6 COMMON	16 SUPPORTIVE	26 TARGET
7 INFORMED	17 REFINED	27 RULING
8 LOCAL	18 VIDEO	28 FACTORS
9 INTERIM	19 NETWORKED	29 OPTIONS
10 VISIBLE	20 PROGRAMMED	30 COMMENTARY

but why not make your own lists ?

```
 10 DIM A$(30,10)
 20 FOR I=1 TO 30
 30 PRINT I
 40 INPUT A$(I)
 50 CLS
 60 NEXT I
100 FOR I=1 TO 21 STEP 10
110 LET B$=A$(I+INT (10*RND))+" "
120 LET B$=B$( TO LEN B$-1)
130 IF B$(LEN B$)=" " THEN GOTO 120
140 PRINT B$;" ";
150 NEXT I
```

```
16Ø PRINT
17Ø PRINT "ANOTHER ?"
18Ø IF INKEY$="" THEN GOTO 18Ø
19Ø CLS

2ØØ IF INKEY$="Y" THEN GOTO 1ØØ
21Ø STOP
22Ø SAVE "BUZZ"
23Ø GOTO 1ØØ
```

As well as being fun to use, this program also illustrates one way of storing fixed data in a ZX81 program when space is limited, and the use of a SAVE statement within a program.

Bob-Sleigh 1K

How good are you at steering a bob-sleigh down a twisting course ? Well, here's your chance to find out. Use keys 5 & 8 to steer, but be careful because the course gradually gets narrower!

```
 1Ø SLOW
 2Ø LET A=5
 3Ø LET B=12
 4Ø LET E=15

1ØØ FOR X=8 TO 3 STEP -.Ø3
11Ø PRINT AT E,B;" "
12Ø SCROLL
13Ø LET C=INT (A+A*SIN (X*A))
14Ø LET D$=INKEY$
15Ø LET B=B-(1 AND D$="5")+(1 AND D$="8")
16Ø PRINT AT E,C;"■"
17Ø PRINT AT E,C+X;"■"
18Ø PRINT AT E,B;"V"
19Ø IF B<=C OR B>=C+X THEN GOTO 3ØØ

2ØØ NEXT X
21Ø PRINT AT 16,C;"END"
22Ø STOP
3ØØ PRINT AT E,B;"CRASH"
```

This program features the SCROLL command, used here to give a 'moving' display. The other feature that may be of interest is the way lines 140 & 150 control the variable B (the position of the bob sleigh) in response to keys 5 and 8; pressing key 5 decrements B each time line 150 is executed, pressing key 8 increments it.

Recipes 1K

The entire membership of the local ZX81 club is coming for dinner, but your favourite recipe for Haggis Dip is only for three servings. Don't worry ! This program will quickly scale the quantities in any recipe to cater for any number of visiting gourmets. Just enter the number of mouths the recipe was written for, the number you have to feed, then each item, and the scaled quantities will be quickly computed.

The quantity of each item must appear first on each line entered, before the description, and can be entered as a decimal number or as a simple fraction e.g;

```
2.5 LB BLACK PUDDING
1/4 CUP OF SALT
```

The results are rounded to one decimal place before printing. To end, enter a null line (just press ENTER (or NEW LINE)).

```
10 PRINT "RECIPE WAS FOR ";
20 INPUT R
30 PRINT R
40 PRINT "NUMBER TO FEED ";
50 INPUT F
60 PRINT F
70 PRINT AT 3,0;"ITEMS;"
80 PRINT
100 INPUT I$
110 IF I$="" THEN STOP
120 FOR C=1 TO LEN I$
130 IF I$(C)<>"/" AND I$(C)<"." OR I$(C)>"9" THEN GOTO 150
140 NEXT C
150 LET V=1
160 IF C>1 THEN LET V=F/R*VAL I$( TO C-1)
170 PRINT INT (.5+V*10)/10;(I$+" ")(C TO )
180 GOTO 100
```

Of course, you don't have to stick to food. The program can be easily modified to scale any type of component list.

If you've got a ZX printer, why not modify the program to give a permanent record of the new quantities ?

SAM 1K

How many enemy aircraft can you shoot down as they fly across the screen ? Use key "F" to fire your Surface to Air Missile, and "BREAK" if you want to stop before all hundred enemy aircraft have passed.

```
 5 SLOW
1Ø LET A=Ø
15 LET G=1ØØ
2Ø LET H=2Ø
25 LET I=1
3Ø FOR B=I TO G
35 LET C=I+INT (5*RND)
4Ø LET D=1Ø
45 PRINT AT D,D;"/"
5Ø PAUSE G
55 FOR E=I TO H
6Ø PRINT AT D,H-D;" "
65 LET D=D-(I AND (INKEY$ ="F" OR D<>1Ø) AND D)
7Ø PRINT AT D,H-D;"/" AND D
75 PRINT AT C,E;" >"
8Ø IF D<>C OR H-I-D<>E THEN GOTO G
85 PRINT AT C,E;" *"
9Ø LET A=A+I
95 LET E=H
1ØØ NEXT E
1Ø5 PRINT AT H,I;A;" HITS OUT OF ";B;AT C,E;"  ";AT D,H-D;"  "
11Ø NEXT B
```

Note the extensive use of the variables G,H and I in place of numeric constants to reduce the program storage requirements. For example, line 55 takes only 11 bytes instead of the 25 bytes which would be taken by the equivalent;

```
55 FOR E=1 TO 2Ø
```

We could have saved even more by, for example, writing lines 10 to 25 as;

```
1Ø LET I=PI/PI
15 LET A=I-I
2Ø LET H=CODE "="
25 LET G=H*CODE "■"
```

But the program works as it stands, so why complicate things ?

Also of interest is line 65, which decrements D each time it is executed, but only down to zero, and only after key F has been pressed.

Cambridge Crypto 1K

This program is essential for spies from Cambridge, Omsk, or any large multinational as it can translate any message into an almost unbreakable code. To break the code, an enemy would have to know the key used for the particular message, as well as the algorithm used by the ZX81 to generate the Random numbers.

```
10 PRINT "ENTER;"
20 PRINT "  E TO ENCODE, D TO DECODE;"
30 INPUT E$
40 IF E$<>"E" AND E$<>"D" THEN GOTO 30
50 PRINT E$
60 PRINT "  KEY (1-65535)"
70 INPUT K
80 IF K>65535 THEN GOTO 70
90 RAND K

100 CLS
110 PRINT "ENTER MESSAGE"
120 INPUT I$
130 PRINT I$
140 FOR A=1 TO LEN I$
150 LET C=CODE I$(A)
160 IF C<12 OR C>63 THEN GOTO 210
170 LET X=INT (52*RND)
180 IF E$="E" THEN LET X=-X
190 LET C=C+X

200 LET C=C+(52 AND C<12)-(52 AND C>63)
210 PRINT CHR$ C;
220 NEXT A
```

The key is used to set the start of the ZX81's random number sequence in line 90. Lines 150-210 then translate each character of the message except for spaces, graphic characters and ZX81 command words. Line 170 generates a different offset for each character in the message.

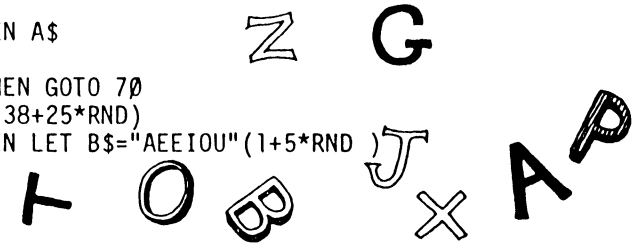
CR?S??ORD 1K

Are you a crossword addict ? Or are you trying to think of a name for your latest piece of software ? Then this program will help you. Enter a word (or mnemonic) with the doubtful letters represented by question marks '?', and the program will display a list of 'words', by replacing the question marks with randomly generated characters. When the screen is full, the program will stop with the report 5/70. Enter CONT for more variations.

But be careful who you allow to use this program.
Although egotistic children will enjoy simply seeing variations

on their name, others have been known to use it to 'accidentally' display rude words !

```
10 INPUT A$
20 FOR C=1 TO LEN A$
30 LET B$=A$(C)
40 IF B$<>"?" THEN GOTO 70
50 LET B$=CHR$ (38+25*RND)
60 IF RND<.3 THEN LET B$="AEEIOU"(1+5*RND )
70 PRINT B$;
80 NEXT C
90 GOTO 20
```

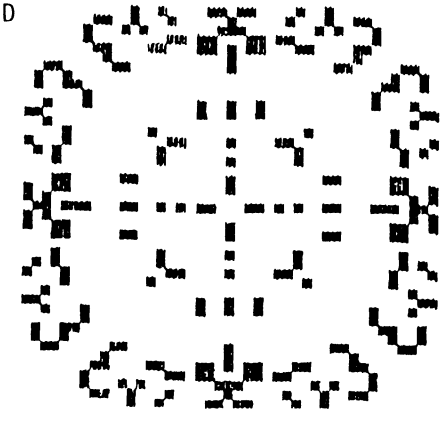


Line 60 allows - in a rather crude way - for the fact that vowels occur frequently in English words. Try improving the program so that it more accurately mimics the frequency with which individual letters occur. Alternatively (and this is more difficult than it may seem at first), re-write it so that it generates the missing letters in alphabetic order, rather than at random.

Starburst 1K

This program draws an endless sequence of symmetrical patterns, pausing for applause after each one.

```
10 LET A=CODE "+"
20 LET B=NOT A
30 LET C=B
40 FOR D=C TO A+A*PI*RND
50 LET E=PI*PI*RND
60 LET B=B+E*RND
70 LET C=C+E*RND
80 LET B=B AND B<=A
90 LET C=C AND C<=A

100 PLOT A+B,A+C
110 PLOT A+B,A-C
120 PLOT A-B,A+C
130 PLOT A-B,A-C
140 PLOT A+C,A+B
150 PLOT A+C,A-B
160 PLOT A-C,A+B
170 PLOT A-C,A-B
180 NEXT D
190 PAUSE CODE "■"

200 CLS
210 RUN
```



All Change 1K

This program is built around a useful Machine Language routine which changes all occurences of one character in the display for another, almost instantaneously.

As listed, it will change any * symbol on the screen to a black square, but this can be altered by adding POKE statements after line 6Ø ;

POKE the code of the character you want changed into 16522.
POKE the code of the new character into location 16535.

```
 1 REM 123456789Ø123456789Ø1234
1Ø LET A$="2AØC4ØED5B1Ø4Ø3E1723EBA7ED52C819EBBE2ØF5368Ø18F1"
2Ø LET A=16514
3Ø FOR B=1 TO LEN A$-1 STEP 2
4Ø POKE A,16*CODE A$(B)+CODE A$(B+1)-476
5Ø LET A=A+1
6Ø NEXT B

1ØØ FOR A=3 TO 18
11Ø PRINT AT A,A;"-*-*-*-*-*-"
12Ø NEXT A
13Ø RAND USR 16514
14Ø FOR A=1 TO 1ØØ
15Ø NEXT A
16Ø CLS
17Ø GOTO 1ØØ
```

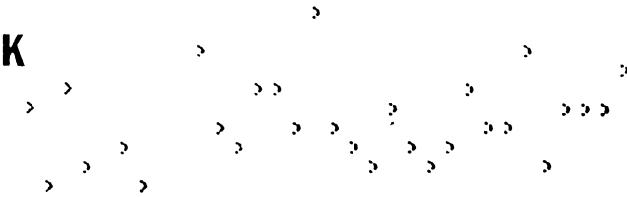
Lines 100 to 170 provide a demonstration of the routine's capabilities, note that RAND USR is the shortest way of using USR if you don't want a result.

Once lines 10 to 60 have been entered and RUN, the machine code is stored in line 1 (which is why it looks funny when listed), and lines 10 to 60 can then be deleted and the variables A,B.A$ CLEARed to make more room for your BASIC program if you wish.

For those interested, the Machine Code routine used is;

```
4Ø82  2A ØC 4Ø          LD HL (4ØØC)    D-FILE
4Ø85  ED 5B 1Ø 4Ø       LD DE (4Ø1Ø)    VARS
4Ø89  3E 17             LD A 17         "*"
4Ø8B  23          LOOP: ┌INC HL
4Ø8C  EB                │EX DE HL       Return
4Ø8D  A7                │AND A           if HL
4Ø8E  ED 52             │SBC HL DE         equals VARS
4Ø9Ø  C8                │RET Z
4Ø91  19                │ADD HL DE
4Ø92  EB                │EX HL DE
4Ø93  BE                │CP (HL)        Is char "*"?
4Ø94  2Ø F5             ├JRNZ LOOP      Jump if not
4Ø96  36 8Ø             │LD (HL) 8Ø     Change it to " "
4Ø98  18 F1             └JR LOOP        & back for next
```

Birds 1K



This is a banal program used to illustrate a useful machine code routine; a horizontal (left to right) scroll.

Each time USR 16514 is executed, everything on the screen is shifted one column to the right, and the first column filled with spaces. The routine won't - however - lengthen a line which is shorter than 32 characters, hence lines 100-130 of the demonstration which fill the first 11 lines of the display with 32 spaces each.

Once the program has been RUN once, lines 10-60 can be deleted, to make room for more useful statements.

```
  1 REM 123456789Ø123456789Ø
 1Ø LET A$="Ø176172AØC4Ø16ØØ237EB928Ø4725718F71ØF3C9"
 2Ø LET A=16514
 3Ø FOR B=1 TO LEN A$-1 STEP 2
 4Ø POKE A,16*CODE A$(B)+CODE A$(B+1)-476
 5Ø LET A=A+1
 6Ø NEXT B

1ØØ DIM A$(32)
11Ø FOR A=1 TO 11
12Ø PRINT A$
13Ø NEXT A

2ØØ FOR A=1 TO 1ØØ
21Ø PRINT AT 1Ø*RND,Ø;">"
22Ø LET B=USR 16514
23Ø NEXT A
```

The machine code routine used is, in assembly language form;

```
4Ø82  Ø1 76 17   LD BC 1776      Start of Disp file
4Ø85  2A ØC 4Ø   LD HL (4ØØC)
4Ø88  16 ØØ      LD D Ø
4Ø8A  23         INC HL          Get
4Ø8B  7E         LD A (HL)        next char
4Ø8C  B9         CP C            Jump if it
4Ø8D  28 Ø4      JR Z 4Ø94        is NEWLINE
4Ø8F  72         LD (HL) D       Disp old char
4Ø91  57         LD D A          Save new one
4Ø92  18 F7      JR 4Ø8A         & round again.
4Ø94  1Ø F3      DJNZ 4Ø88       Quit if
4Ø96  C9         RET              last NEWLINE
```

The value in location 4084 - given here as 17 hex- is one more than the number of lines to be scrolled. So for example if it were made 09, then only the top 8 lines of the display would scroll sideways.

Alien Attack 1K

A fleet of nearly 400 alien space ships is approaching Earth. How many can you blast before the game ends or - worse - you collide with one of them ?

Use keys 6 and 7 to move your ship down or up the screen,and key F to fire. Your score will be shown when the game ends.

The program must be entered in two stages.

First enter;

```
 1 REM 12345678901234567890123456
10 LET A$="2A0E404E06000C90176192A1040160002B7EB92804725718F71
   0F3C9"
20 LET A=16514
30 FOR B=1 TO LEN A$-1 STEP 2
40 POKE A,16*CODE A$(B)+CODE A$(B+1)-476
50 LET A=A+1
60 NEXT B
```

Check line 10 carefully (there should be 54 characters betwen the quotation marks) , then RUN it.

Next, delete all of the lines except for line 1, which will now look very strange. Then enter;

```
10 LET S=NOT PI
20 LET A=CODE "<"
30 LET B=A
50 LET Z=S

100 FOR D=S TO A*A
110 PRINT AT B,Z;
120 IF USR 16514=A THEN GOTO 300
130 PRINT ">";
140 IF INKEY$="F" THEN GOSUB 500
190 LET B=B-(INKEY$="7" AND B)+(INKEY$="6" AND B<A)

200 PRINT AT RND*A,17;"<"
210 LET C=USR 16521
220 NEXT D

300 PRINT "*";S;P

500 FOR C=Z TO PI
510 IF USR 16514=A THEN LET S=S+1
520 PRINT "-";
530 NEXT C
540 PRINT AT B,Z;">   "
550 RETURN
```

You can then RUN and SAVE the program as normally.

Several tricks have been used to save space and to increase the speed of the game as much as possible. Of particular interest may be line 190, which moves your ship's position (B) according to operation of the keys 6 & 7 but always keeping it within the limits 0 and 19. Also, including the previously unused variable P in line 300 halts the game without needing a separate STOP line.

Two machine code routines are used. The first, starting at 16514, returns the code of the character at the current print position. It is similar to, but faster than;

```
PEEK (PEEK 16398+256*PEEK 16399)
```

In assembly language form, it is;

```
4Ø82  2A ØE 4Ø    LD HL (4ØØE)
4Ø85  4E          LD C (HL)
4Ø86  Ø6 ØØ       LD B Ø
4Ø88  C9          RET
```

The second routine, starting at 16521, is a horizontal (right to left) scroll;

```
4Ø89  Ø1 76 19    LD BC 1976
4Ø8C  2A 1Ø 4Ø    LD HL (4Ø1Ø)     VARS
4Ø8F  16 ØØ   ┌─→ LD D Ø
4Ø91  2B      │ ┌→ DEC HL          Get previous
4Ø92  7E      │ │  LD A (HL)        character
4Ø93  B9      │ │  CP C            Jump if it was
4Ø94  28 Ø4   │ │ ┌ JR Z 4Ø9C       NEWLINE
4Ø96  72      │ │ │ LD (HL) D      Replace it
4Ø98  57      │ │ │ LD D A
4Ø9A  18 F7   │ │ └ JR 4Ø91        & round again
4Ø9C  1Ø F3   └─│──→ DJNZ 4Ø8F     Quit after 25
4Ø9E  C9        │    RET             NEWLINES
                └──
```

Altering the value in the third byte of this routine (given here as 19 hex) will affect the number of lines of the display which are scrolled sideways.

Carrier 1K

By Lee Gentry

Fancy your chances as a Navy pilot ? Perhaps playing this simple simulation will bring you down to earth.

Use keys '6' and '7' to control the height of your plane, and try to land on the centre of the carrier deck without hitting the superstructure or landing so far forward that your plane is liable to topple over the front of the ship.

```
 1Ø LET H=NOT PI
 2Ø LET C=H
 3Ø LET S=H
 4Ø IF C>=VAL "32" THEN LET C=NOT PI
 5Ø IF S>=VAL "32" THEN LET S=NOT PI
 6Ø CLS
 7Ø PRINT AT VAL "17",C;" ";TAB C;"■";TAB C;"▦▦▦▦▦▦▦▦";TAB C;
    "▬▬▬▬";AT H,S;"▚▞";
 8Ø IF H>=VAL "17" THEN GOSUB VAL "2Ø*H-19Ø"
 9Ø LET H=H+(INKEY$="6")-(INKEY$="7" AND H)
1ØØ LET S=S+VAL "3"
11Ø LET C=C+VAL "2"
12Ø GOTO VAL "4Ø"
15Ø IF (S>C+PI/PI OR S<C-PI/PI) AND H=VAL "17" THEN RETURN
17Ø IF (S=C+INT PI OR S=C+VAL "4") AND H=VAL "18" THEN GOTO
    VAL "22Ø"
18Ø IF (S<>C-PI/PI AND S<>C+VAL "5") AND H=VAL "18" THEN RETURN
19Ø IF (S<>C-VAL "2" AND S<>C+VAL "6") AND H=VAL "19" THEN RETURN
2ØØ PRINT "CRASH";Q
21Ø PRINT "DITCHES";Q
22Ø PRINT AT PI,NOT PI;"SAFELY DOWN"
```

The 5 strings between quotation marks in line 7Ø are entered as;

"space ,graphics/W"
"space ,inverse-space"
"inverse plus sign (6 times),graphics/1"
"graphics/2 , graphics/7 (three times), graphics/1"
"graphics/W, graphics/F, graphics/4"

The speeds of the plane and the carrier can be altered by changing lines 100,110 or - if you have more than 1K RAM, the whole game can be speeded up by replacing the VAL statements by proper - simple - integers.

Calendar

```
SU MO TU WE TH FR SA
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

By Lee Gentry **1K**

This program will either display on the TV screen a calendar for any month from January 1901 to December 1999, or it will print out on the ZX printer a twelve month calendar for any year 1901 to 1999.

```
 10 LET B=NOT PI
 20 LET A$="144025036l462"
 30 PRINT "YEAR";TAB B;"MONTH";TAB B;"PAPER"
 40 INPUT Y
 50 INPUT M
 60 INPUT I$
 70 LET Y=Y-VAL "1900"
 80 IF I$="P" THEN FOR M=U TO X*INT PI
 90 IF Y=X*INT (Y/X) THEN LET A$="034025036l462"
100 LET A=X*Y+Y+X*VAL A$(M)
110 LET W=U+INT ((A-V*INT (A/V))/X)
120 PRINT TAB NOT PI;"    ";M;TAB NOT PI;"SU MO TU WE TH FR SA"
130 PRINT TAB W*X-W-X+U;I;
140 LET I=I+U
150 LET W=W+U
160 IF W=X+X THEN LET W=U
170 IF A$(M+U)<A$(M) THEN LET B=VAL "7"
180 IF I<=V+VAL A$(M+U)-VAL A$(M)+B THEN GOTO VAL "130"
190 LET I=U
200 LET B=NOT PI
210 IF I$<>"P" THEN STOP
220 COPY
230 CLS
240 NEXT M
```

Don't RUN the program, but enter the following direct commands;

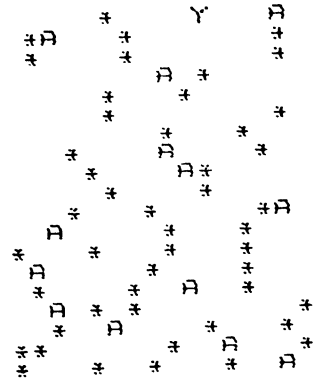
```
LET U=1
LET I=1
LET X=4
LET V=28
FAST
GOTO 10
```

It will then display

```
YEAR
MONTH
PAPER
```

You should then enter three separate lines; first the year, then the month (even if you want a whole year's calendar printed out), and finally enter 'P' to get a printout or anything else to display just one month.

Swarm By Lee Gentry 1K



You are a strange 'Y' shaped being living at the top of the screen, and can move from side to side by pressing the '5' and '8' keys on your ZX81 based flight controller. Coming up at you is a swarm of other beings. Most of them look like '*'s and are poisonous so should be avoided. But a few (which look uncannily like the letter 'A') are your favourite food so you should try to intercept them before they escape off the top of the screen. As you get deeper into the swarm, you will find more and more of the poisonous beings.

Can you survive the *SWARM ?*

First enter and RUN;

```
 1 REM 1234567
10 LET A$="2AØE4Ø4EØ6ØØC9"
20 LET A=16514
30 FOR B=1 TO 13 STEP 2
40 POKE A,16*CODE A$(B)+CODE A$(B+1)-476
50 LET A=A+1
60 NEXT B
```

then (without entering NEW) enter and RUN;

```
10 LET S=NOT PI
20 LET C=VAL"1Ø"
30 FOR N=PI/PI TO VAL "7" STEP VAL "1/4Ø"
40 LET H=NOT PI
50 PRINT AT PI-PI,VAL "21";S;AT PI-PI,C;
60 IF USR VAL "16514"=CODE "A" THEN LET S=S+PI/PI
70 IF USR VAL "16514"=CODE "*" THEN GOTO VAL "25Ø"
80 PRINT "Y"
90 LET C=C-(INKEY$="5" AND C)+(INKEY$="8" AND C<VAL "19")
100 PRINT AT CODE "(",RND*VAL "19"/INT N+H;"*"
110 LET H=H+VAL "19"/INT N
120 IF H<VAL "19" THEN GOTO VAL "1ØØ"
130 PRINT AT CODE "(",RND*VAL "19";"A"
140 SCROLL
150 NEXT N
250 PRINT "X";S;P
```

This program uses the machine code routine from 'Alien Attack' to get the code of the character in the current print position.

Several tricks have been used to squeeze the program into 1K RAM, if you have more than 1K then try undoing them (e,g, replacing VAL"16514" by the plain 16514, and CODE "(" by 16, and PI/PI by 1). and you will be suprised at how much faster the program will run.

Bulbs By Lee Gentry 1K

This program shows a row of 'light bulbs' moving across the screen through 10 positions. You have to try to 'hit' the leading bulb by pressing the key '1' to '0' corresponding to its position on the screen; if you succeed then that bulb goes out, but if you miss then another bulb is added to the moving row. To win, you must wipe out all of the bulbs. If your aim is so abysmal that you end up with 10 'bulbs' on the screen then you lose !

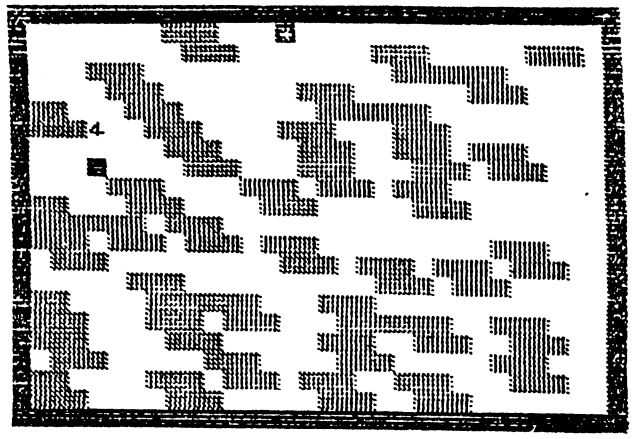
When it starts, the program asks for your 'Skill'. Enter a number between 0 and 20; the lower the number the faster the bulbs will move.

```
 10 LET L=INT PI
 20 PRINT "SKILL ";
 30 INPUT S
 40 PRINT S
 50 LET B=-PI/PI
 60 LET A=L*L+B
 70 IF B=NOT PI THEN GOTO 110
 80 IF A>30 THEN LET A=2
 90 IF B>30 THEN LET B=2
100 PRINT AT 3,B;" "
110 PRINT AT 3,A;"O"
120 LET C=NOT PI
130 LET C=C+PI/PI
140 IF C<S THEN GOTO 130
150 LET G=CODE INKEY$
160 LET B=B+INT PI-30*INT ((B+PI)/30)
170 IF G<>0 THEN GOSUB 200
180 LET A=A+INT PI
190 GOTO 70

200 IF G=28 THEN LET G=38
210 IF INT (G*3-A)=85 THEN GOTO 260
220 LET L=L+PI/PI
230 LET B=B-INT PI
240 IF L>PI*PI THEN PRINT AT PI,B;"O";TAB 0;"YOU LOSE";Q
250 RETURN
260 LET L=L-PI/PI
270 PRINT AT PI,B;" "
280 LET B=B+INT PI
290 IF L=NOT PI THEN PRINT "YOU WIN";Q
300 RETURN
```

This program is noteworthy for the number of ways it uses 'PI' to save memory space.

Dragon!

By Lee Gentry



DRAGON is a game of cunning and quick thinking played on a square plateau. You can travel anywhere on the plateau by using keys 5,6,7 & 8, but be careful not to fall over the edge, as that is fatal !

The plateau is dotted with grey woods and swamps, which will slow you down if you try to go through them, and somewhere lies a pile of treasure, represented by a single number, which you must collect. When you have collected one pile another appears.

But, there is also a hungry Dragon on the plateau, and he is after you. Like you he is hindered by the swamps and more so by the woods, but he is so eager to get you that he destroys the woods as he passes through them, reducing your cover. Being a very mean type of Dragon, he will also bury any treasure that he finds, so that you can no longer see it and will have to locate it by memory and luck.

How long can you survive the Dragon ?

```
 5 LET ZZ=Ø
1Ø LET S=Ø
15 GOSUB 75Ø
2Ø LET B=Ø
25 LET D=B
3Ø LET G=B
35 LET H=B
4Ø LET P=B
45 LET Q=B
5Ø LET E=2
55 LET F=3Ø
6Ø LET W=INT (RND*2Ø+1)
65 LET Z=INT (RND*3Ø+1)
7Ø IF A(W+1,Z+1)<>Ø THEN GOTO 6Ø
75 IF ABS (W-E)<3 AND ABS (Z-F)<3 THEN GOTO 6Ø
8Ø SLOW
```

```
 85 PRINT AT W,Z;"■"
 9Ø LET A(W+1,Z+1)=5
 95 GOSUB 55Ø

1ØØ LET B=B+1
11Ø IF D<Ø THEN GOTO 3ØØ
12Ø LET C=Ø
13Ø IF INKEY$="" THEN LET C=C+1
14Ø IF C=1Ø THEN GOTO 3ØØ
15Ø IF INKEY$="" THEN GOTO 13Ø
16Ø LET A(W+1,Z+1)=P
17Ø PRINT AT W,Z;" "
18Ø IF P=1 THEN PRINT AT W,Z;"▩"
19Ø LET W=W+(1 AND INKEY$="6")-(1 AND INKEY$="7")
2ØØ LET Z=Z+(1 AND INKEY$="8")-(1 AND INKEY$="5")
21Ø IF A(W+1,Z+1)=4 OR A(W+1,Z+1)=1Ø THEN GOTO 11ØØ
22Ø IF W=X AND Z=Y THEN GOSUB 65Ø
23Ø IF A(W+1,Z+1)=2 THEN LET D=-1
24Ø IF A(W+1,Z+1)=1 THEN LET D=-2
25Ø LET P=A(W+1,Z+1)
26Ø LET A(W+1,Z+1)=5
27Ø PRINT AT W,Z;"■"
3ØØ LET D=D+1
31Ø IF B<Ø THEN GOTO 1ØØ
32Ø IF ABS (E-1-W)>ABS (F-1-Z) THEN LET G=1
33Ø IF G=Ø THEN LET H=1
34Ø IF G>Ø.5 AND E-1>W THEN LET G=-1
35Ø IF H>.5 AND F-1>Z THEN LET H=-1
36Ø LET A(E,F)=Q
37Ø IF Q=1 THEN LET A(E,F)=Ø
38Ø PRINT AT E-1,F-1;" "
39Ø LET E=E+G
4ØØ LET F=F+H
41Ø PRINT AT E-1,F-1;"▣"
42Ø IF A(E,F)=5 THEN GOTO 11ØØ
43Ø LET B=Ø
44Ø IF A(E,F)=2 THEN LET B=-2
45Ø IF A(E,F)=1 THEN LET B=-3
46Ø LET Q=A(E,F)
47Ø LET A(E,F)=4
48Ø LET G=Ø
49Ø LET H=Ø
5ØØ GOTO 1ØØ

54Ø REM SCORING
55Ø LET X=INT (RND*2Ø+1)
56Ø LET Y=INT (RND*3Ø+1)
57Ø IF ABS (X-E)<3 AND ABS (Y-F)<3 THEN GOTO 55Ø
58Ø LET AA=Ø
59Ø LET K=INT (RND*9+1)
6ØØ PRINT AT X,Y;K
61Ø LET K=-K
```

```
62Ø LET AA=A(X+1,Y+1)
63Ø LET A(X+1,Y+1)=K
64Ø RETURN
65Ø LET S=S-K
66Ø POKE 16418,Ø
67Ø PRINT AT 23,16;"YOUR SCORE = ";S
68Ø POKE 16418,2
69Ø IF A(X+1,Y+2)<9 AND AA=1 THEN LET A(X+1,Y+2)=1
7ØØ IF AA=1 THEN PRINT AT X,Y;"▒▒"
71Ø IF A(X+1,Y+2)=1Ø THEN PRINT AT X,Y+1;"■"
72Ø LET A(X+1,Y+1)=AA
73Ø GOSUB 55Ø
74Ø RETURN

75Ø REM PREPARATION
76Ø FAST
77Ø CLS
78Ø POKE 16418,Ø
79Ø PRINT AT 23,Ø;"HIGH SCORE =";ZZ;TAB 16;"YOUR SCORE = ";S
8ØØ POKE 16418,2
81Ø DIM A(22,32)
82Ø FOR A=1 TO 4Ø
83Ø LET X=INT (RND*19+2)
84Ø LET Y=INT (RND*27+2)
85Ø IF A(X,Y)=1 OR A(X+1,Y)=1 OR A(X,Y+3)=1 OR A(X+1,Y+3)=1
    THEN GOTO 83Ø
86Ø LET A(X,Y)=1
87Ø LET A(X,Y+1)=1
88Ø LET A(X,Y+2)=1
89Ø LET A(X+1,Y+1)=1
9ØØ LET A(X+1,Y+2)=1
91Ø LET A(X+1,Y+3)=1
92Ø PRINT AT X,Y;"▒▒▒▒"
93Ø PRINT AT X-1,Y-1;"▒▒▒▒"
94Ø NEXT A

95Ø FOR W=Ø TO 31
96Ø IF W>21 THEN GOTO 1Ø2Ø
97Ø PRINT AT W,Ø;"■"
98Ø PRINT AT W,31;"■"
99Ø LET A(W+1,1)=1Ø
1ØØØ LET A(W+1,32)=1Ø
1Ø1Ø LET A(INT (RND*19+2),INT(RND *29+2) =2
1Ø2Ø PRINT AT Ø,W;"■"
1Ø3Ø PRINT AT 21,W;"■"
1Ø4Ø LET A(1,W+1)=1Ø
1Ø5Ø LET A(22,W+1)=1Ø
1Ø6Ø NEXT W
1Ø7Ø RETURN

11ØØ CLS
111Ø IF A(W+1,Z+1)=1Ø THEN GOTO 115Ø
```

```
1120 IF A(W+1,Z+1)=4 THEN PRINT AT 12,0;"IDIOT-YOU WALKED INTO
     THE DRAGON"
1130 PRINT AT 14,5;"HE KILLED YOU"
1140 GOTO 1160
1150 PRINT AT 12,0;"YOU FELL OVER THE CLIFF"
1160 PRINT AT 10,0;"HIGH SCORE = ";ZZ;"  YOU GOT ";S;AT 16,5;
     "YOU ARE DEAD"
1170 IF S>=ZZ THEN PRINT AT 18,1;"**********CHAMPION**********"
1180 IF S>=ZZ THEN LET ZZ=S
1190 PRINT AT 20,0;"ANOTHER TRY ? (Y/N)"
1200 INPUT I$
1210 IF I$="Y" THEN GOTO 10
1220 IF I$<>"N" THEN GOTO 1200
```

Note that AA and ZZ are two-letter variable names, not keywords.

The graphics characters used in the program are;

*inverse * (line 410)*
graphics/A (once in line 180 and three times each in 920,930 twice in line 700).
inverse space (elsewhere).

The program uses the whole 24 lines of the display by POKEing suitable values into location 16418 (lines 660,680,780,800).

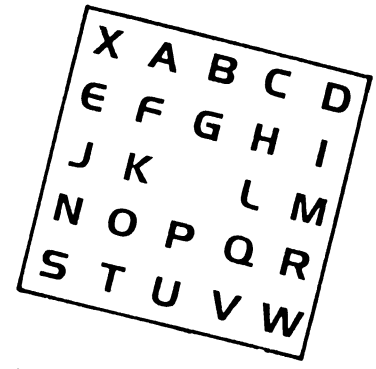
Buzzbox 1K

This complicated program turns your ZX81 into a *very* crude form of musical instrument. Use keys 1 to 9 to play, and listen to the result by connecting an amplifier to the MIC socket (cassette recorders which allow you to monitor what is being recorded are suitable), or you may be able to hear it from your TV set speaker by careful tuning.

```
  5 FAST
 10 GOTO CODE INKEY$
 28 RAND
 29 RAND
 30 RAND
 31 RAND
 32 RAND
 33 RAND
 34 RAND
 35 RAND
 36 RAND
 37 RAND
200 SLOW
210 RUN
```

The sound is produced by lines 5 and 40, which toggle the state of the ZX81's cassette output port. The frequency at which this happens is controlled by lines 10-37, the RAND statements being a convenient way of introducing suitable delays into the loop; see what happens if you substitute other simple statements such as REM.

Mixup



This is a computerised version of the old sliding letter square puzzle, in which you have re-arrange 24 letters in a 5 x 5 board by moving one letter at a time into the empty square.

When the program is run, it first shows you the board with the letters in order. It then scrambles them according to the degree of 'skill' you have chosen, then waits patiently while you try to unscramble the mess.

```
100 DIM Z$(15)
110 LET A$="00-10001-1000100"
120 CLS
130 FOR A=0 TO 23
140 PRINT AT 6+INT (A/5),4+A-5*INT (A/5);CHR$ (38+A)
150 NEXT A
160 LET X=8
170 LET Y=10
180 LET M=1

200 PRINT AT 8,14;"SKILL (1-9)?"
210 LET S=5*CODE INKEY$-140
220 IF S<5 OR S>45 THEN GOTO 210
230 PRINT AT 8,14;Z$

300 LET P=0
310 LET Q=0
320 FOR A=1 TO S
330 LET B=1+4*INT (4*RND)
340 LET U=X+VAL A$(B TO B+1)
350 LET V=Y+VAL A$(B+2 TO B+3)
360 IF U<4 OR U>8 OR V<6 OR V>10 OR (U=P AND V=Q) THEN GOTO 330
```

```
37Ø GOSUB 1000
38Ø NEXT A

4ØØ PRINT AT 7,14;"MOVE ";M;AT 9,14;"WHICH LETTER ?"
41Ø LET B=CODE INKEY$
42Ø IF B<38 OR B>61 THEN GOTO 41Ø
43Ø PRINT AT 7,14;Z$;AT 9,14;Z$

5ØØ FOR A=1 TO 13 STEP 4
51Ø LET U=X+VAL A$(A TO A+1)
52Ø LET V=Y+VAL A$(A+2 TO A+3)
53Ø PRINT AT V,U;
54Ø IF B=PEEK (PEEK 16398+256*PEEK 16399) THEN GOTO 6ØØ
55Ø NEXT A
56Ø GOTO 4ØØ

6ØØ LET M=M+1
61Ø GOSUB 1ØØØ
62Ø FOR A=Ø TO 23
63Ø PRINT AT 6+INT (A/5),4+A-5*(INT(A/5);
64Ø IF PEEK (PEEK 16398+256*PEEK 16399)<> 38+A THEN GOTO 4ØØ
65Ø NEXT A
66Ø PRINT AT 6,14;"**** FINISHED ****";AT 8,16;"IN ";M-1;" MOVES"
67Ø IF M<1.5*S THEN PRINT AT 1Ø,18;"** WELL DONE **"
68Ø STOP

1ØØØ PRINT AT V,U;
1Ø1Ø LET B=PEEK (PEEK 16398+256*PEEK 16399)
1Ø2Ø PRINT " ";AT Y,X;CHR$ B
1Ø3Ø LET P=X
1Ø4Ø LET Q=Y
1Ø5Ø LET X=U
1Ø6Ø LET Y=V
1Ø7Ø RETURN
```

The program works directly with the letters displayed on the screen, PRINTing in the appropriate places, and looking to see what letter is displayed at the current print position by the

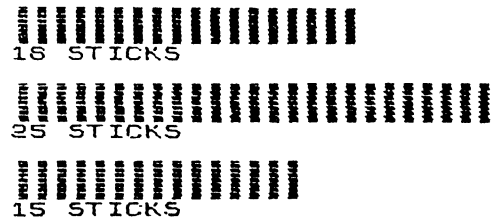
*PEEK (PEEK 16398+256*PEEK 16399)*

statements. A$ holds the horizontal and vertical movements corresponding to the four possible moves from any position.

Why not extend the program to allow you to switch to 'auto pilot' at any stage and let the ZX81 unscramble the pattern ?

Nim

By Lee Gentry



In this version of NIM, you play against the almost unbeatable ZX81 (depending on the starting position, if you play perfectly then you can beat the machine).

For those who don't know the game, it starts with several rows of sticks. You and the computer then take it in turns to remove at least one stick from one of the rows - you must take at least one stick - and the winner is the one who is left with the last one .

```
 1Ø REM PREPARE
 2Ø DIM A(5)
 3Ø DIM A$(1)
 4Ø DIM B$(6,5)
 5Ø DIM C$(5)
 6Ø DIM D$(32)
 7Ø LET D=Ø

1ØØ PRINT AT Ø,Ø;"DO YOU WISH TO CHOOSE THE SETUP?"
11Ø INPUT A$
12Ø IF A$<>"Y" AND A$<>"N" THEN GOTO 11Ø
13Ø LET A=INT (RND*3+3)
14Ø IF A$="Y" THEN PRINT AT Ø,Ø;D$;AT Ø,Ø;"2,3,4,OR 5 PILES ?"
15Ø IF A$="Y" THEN INPUT A
16Ø IF A<2 OR A<>INT A OR A>5 THEN GOTO 13Ø
17Ø PRINT AT Ø,Ø;D$;AT Ø,Ø;A;" PILES"
18Ø FOR B=1 TO A
19Ø LET A(B)=INT (RND*16+1Ø)
2ØØ IF A$="Y" THEN PRINT AT 21,Ø;"PILE ";B;" HOW MANY STICKS
    (1Ø-25)?"
21Ø IF A$="Y" THEN INPUT A(B)
22Ø IF A(B)<1Ø OR A(B)<>INT A(B) OR A(B)>25 THEN GOTO 19Ø
23Ø FOR C=Ø TO A(B)-1
24Ø PRINT AT 4*B-3,C;"■";TAB C;"■"
25Ø NEXT C
26Ø PRINT A(B);" STICKS"
27Ø LET D=D+A(B)
28Ø NEXT B

3ØØ PRINT AT 21,Ø;D$;AT 21,Ø;"WHICH PILE ?"
31Ø INPUT P
32Ø IF P<1 OR P<>INT P OR P>A THEN GOTO 3ØØ
33Ø PRINT AT 21,Ø;"HOW MANY STICKS ?"
34Ø INPUT S
```

```
350 IF S<1 OR S<>INT S OR S>A(P) THEN GOTO 350
360 PRINT AT 21,0;"YOU TOOK ";S;" FROM PILE ";P
370 LET D=D-S
380 LET A(P)=A(P)-S
390 FOR B=A(P) TO A(P)+S
400 PRINT AT 4*P-3,B;" ";TAB B;" "
410 NEXT B
420 PRINT A(P);" "
430 IF D<2 THEN GOTO 900

460 REM COMPUTERS TURN
470 LET S=0
480 LET B$(6)="00000"
490 FOR B=1 TO A
500 LET K=A(B)
510 FOR C=5 TO 1 STEP -1
520 LET B$(B,C)=STR$ (5*(K-2*INT (K/2)))
530 LET K=INT (K/2)
540 LET B$(6,C)=STR$ ((VAL B$(6,C)+VAL B$(B,C))-(10 AND
    B$(6,C)="5" AND B$(B,C)="5"))
550 NEXT C
560 NEXT B
570 IF B$(6)="00000" THEN GOTO 860
580 FOR C=1 TO A
590 FOR K=1 TO 5
600 IF B$(C,K)<>"0" THEN GOTO 640
610 NEXT K
620 IF K<B THEN LET B=K
630 NEXT C
640 LET V=INT (RND*A+1)
650 FOR P=V TO V+A-1
660 LET P=P-A*INT (P/A)+(A AND P=A)
670 LET C$=B$(P)
680 FOR C=B TO 5
690 IF VAL B$(6,C)<>2*INT (VAL B$(6,C)/2) THEN LET C$(C)=
    STR$ (5*(NOT VAL B$(P,C)))
700 NEXT C
710 IF C$>B$(P) THEN GOTO 660
720 LET K$=STR$ (VAL B$(P)-VAL C$)
730 FOR C=LEN K$ TO 1 STEP -1
740 IF K$(C)="5" OR K$(C)="9" THEN LET S=S+2**(LEN N$-C)
750 NEXT C
760 PRINT AT 21,0;"I TAKE ";S;" FROM PILE ";P
770 LET A(P)=A(P)-S
780 LET D=D-S
790 FOR B=A(P) TO A(P)+S
800 PRINT AT 4*P-3;B;" ";TAB B;" "
810 NEXT B
820 PRINT A(P);" "
830 IF D>1 THEN GOTO 300
840 LET D=NOT D
850 GOTO 900
```

```
86Ø LET P=INT (RND*A+1)
87Ø IF A(P)<1 THEN GOTO 86Ø
88Ø LET S=INT (RND*A(P)+1)
89Ø GOTO 75Ø

9ØØ CLS
91Ø PRINT AT 8,9;"**********";AT 1Ø,9;"*** I WIN ***";
    AT 1Ø,11;(" YOU WIN " AND D<.5);AT 12,9;"*********"
```

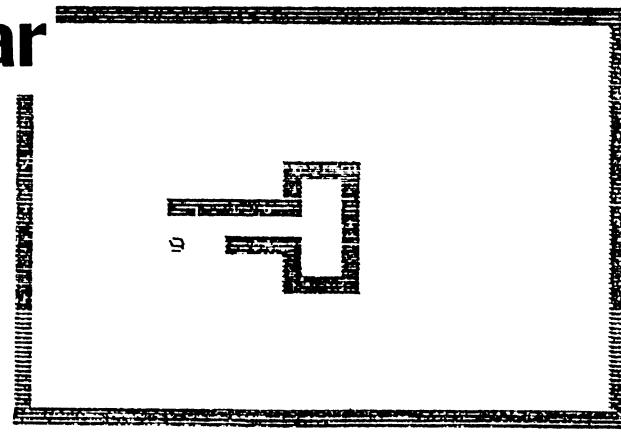
The algorithm used by the computer to calculate its move is interesting; it first represents the number of sticks in each row as a binary number, then adds the number of 'ones' in the corresponding positions of each binary number;

E.g; 1st. row; 5 sticks ; binary 1 Ø 1
2nd. " 6 " " 1 1 Ø
3rd. " 4 " " 1 Ø Ø
4th. " 3 " " Ø 1 1
3 2 2

It then sees if it can - by taking a number of sticks from one pile - create a 'safe' position, which is one in which the number of 'ones' in each column is even. In the example shown above, taking 4 sticks from the first, second or third row would achieve this.

Caterpillar

By Lee Gentry



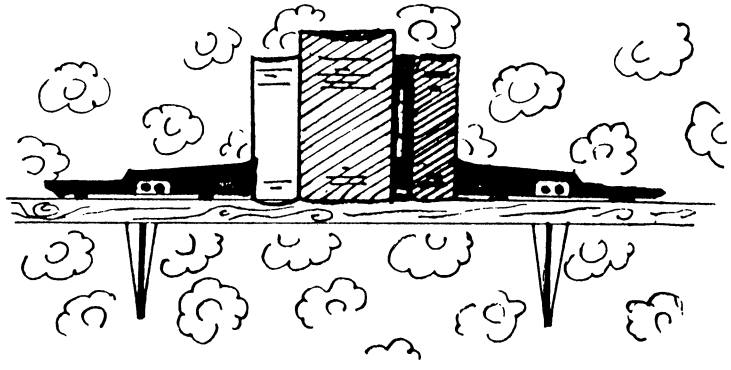
You are a very hungry caterpillar crawling across the screen looking for food. You can change direction by using the keys 5,6,7 & 8, but if you hit the edge of the screen, or move across yourself, or double back on yourself, then you die! The 'food' you are looking for is represented by a number somewhere on the screen, you 'eat' it by moving your head over it, and when you have eaten one piece, another will appear. *But, every time you eat you grow!* So as the game progresses your body gets longer and longer, so it gets more and more difficult to avoid eating yourself. When you do finally expire, a score is given .

```
  1 REM 1234567
  2 LET A$="2AØE4Ø4EØ6ØØC9"
  3 LET A=16514
  4 FOR B=1 TO 13 STEP 2
  5 POKE A,16*CODE A$(B)+CODE A$(B+1)-476
  6 LET A=A+1
  7 NEXT B

 1Ø REM PREPARE
 2Ø LET S$="44"
 25 LET H=4
 3Ø LET U=1Ø
 35 LET V=15
 4Ø LET J=Ø
 45 LET B=4
 5Ø LET S=2
 6Ø FOR A=Ø TO 31
 7Ø IF A<22 THEN PRINT AT A,Ø;"■";TAB 31;"■"
 8Ø PRINT AT Ø,A;"■";AT 21,A;"■"
 9Ø NEXT A

1ØØ LET X=U
11Ø LET Y=V
12Ø PRINT AT X,Y;"■"
13Ø GOSUB 5ØØ
15Ø REM MOVE
16Ø LET F$=INKEY$
17Ø IF F$="" THEN LET B=H
18Ø IF F$="" THEN GOTO 2ØØ
19Ø LET B=VAL F$-4
2ØØ LET X=X+(1 AND B=2)-(1 AND B=3)
21Ø LET Y=Y+(1 AND B=4)-(1 AND B=1)
22Ø LET H=B
23Ø LET B=Ø
24Ø PRINT AT X,Y;
25Ø IF USR 16514>=1 THEN GOSUB 4ØØ
26Ø PRINT "■"
27Ø IF J>=1 THEN GOTO 33Ø

275 REM DELETE
28Ø PRINT AT U,V;" "
29Ø LET S$(1 TO S)=S$(2 TO S)+STR$ H
3ØØ IF S$(2)="2" OR S$(2)="3" THEN LET U=U-(2*VAL S$(2)-S)
31Ø IF S$(2)="1" OR S$(2)="4" THEN LET V=V+(2*VAL S$(2)-S)/3
32Ø IF J<1 THEN GOTO 15Ø
33Ø LET J=J-1
34Ø LET S$(S-J)=STR$ H
35Ø GOTO 15Ø

4ØØ REM SCORE
41Ø IF USR 16514=128 THEN PRINT "*";LEN S$-2;Q
42Ø FOR A=1 TO K
43Ø LET S$=S$+" "
```

```
44Ø NEXT A
45Ø LET J=J+K
46Ø LET S=S+K
47Ø PRINT AT W,Z;"■"
5ØØ LET K=INT (RND*9+1)
51Ø LET W=INT (RND*2Ø+1)
52Ø LET Z=INT (RND*3Ø+1)
53Ø PRINT AT W,Z;
54Ø IF USR 16514=128 THEN GOTO 51Ø
55Ø PRINT K
56Ø RETURN
```

The first seven lines set up a machine code routine which gives the code of the character at the current Print position, after the program has been Run line 1 will look strange, but don't worry.

Chapter 5



Applications

Application Notes

It seems that most of the software written for the ZX81 is for games of one form or another. This isn't necessarily a bad thing, writing games programs is an enjoyable way of learning to program a computer and the simple rules and structure of most games means that the program shouldn't be too complicated and so you stand a fair chance of getting it to run . And - after all - games programs are fun to run ! But let us not forget that the ZX81 is more powerful in many ways than the first generation of business computers, and can do many useful jobs in a business or engineering enviroment. And it is not limited to traditional calculating and accounting applications; if fitted with a suitable I/O port is makes an ideal controller for laboratory use in all sciences.

Some examples of such 'useful' programs are given in this chapter - and the reader may find an application for them as they stand, or he may want to tailor them to his specific needs. First, however, it is worthwhile reviewing the strengths and the weaknesses of the ZX81 for 'real' applications.

MATHEMATICAL CAPABILITIES

These are generally very good, notably the ZX81 has a full set of trigonometrical functions, although it doesn't allow the user to define his own functions and there are no matrix arithmetic instructions.

CHARACTER HANDLING

The ZX81 can manipulate characters and strings as least as well as any other computer.

DISPLAY

Most 'proper' computer terminals can display 80 or more characters per line. The ZX81's limit of 32 characters per line is obviously a disadvantage but not a serious one except for word processing applications.

KEYBOARD

No one can pretend that the ZX81 keyboard is ergonomically suited to continuous use ! If you are going to use the machine a lot it is worth buying one of the full sized add-on keyboards now available for the ZX81.

LANGUAGE

BASIC is not an ideal language for large and/or complex programs. It is all too easy to write a program in BASIC that is almost incomprehensible - even to the author - and if you can't understand your program how can you be sure that it will *always* work ?

SECURITY

In this imperfect world, there is always a chance that your program or data can get corrupted while in the computer's memory or while being transferred to and from tape. Large computers have special circuitry to detect errors occuring in the machine itself, and even most microcomputers check to see that the data read from tape is error-free. Unfortunately, the ZX81 does neither of these things and so you cannot really use it for serious accountancy work unless additional routines are included within the program to check data and the program itself for validity as often as required.

SPEED

Reasonable. Not in the Cray supercomputer class, but adequate for most applications.

RANDOM ACCESS MEMORY

With the 16K RAM pack, and allowing for the average size of program, you are left with enough room for about 10000 characters or 2000 floating point numbers. In practical terms, this is equal to about 125 names and addresses. 48K RAM packs are now available which can give you space for over 40000 characters (500 names and addresses).

BULK MEMORY

The most serious disadvantage of the ZX81 for many business applications is that no disc drives are available ; at least not at the time of writing. Most commercial data processing consists of extracting or altering just a small part of a large data file, and the disc drive is ideally for suited for this.

With the ZX81, however, using a domestic cassette recorder limits you to handling data files no longer than can fit into RAM. Although you could, theoretically, break a large file up into RAM sized chunks and deal with each part separately, the difficulties of having to keep starting, stopping and rewinding the recorder, as well as changing tapes, and also the long access time (it takes about 7 minutes to load or save a 16K file, once you have located it on the tape), really make it impracticable except perhaps in very special circumstances.

Overall then, it seems that the ZX81 is best suited to applications which need to input or output only a small amount of data at a time, and which do not use data files larger than about 2000 numbers or 10000 characters. Also, the programmer may need to take special precautions to safeguard against corruption of the program or the stored data. Otherwise, the only limits to using the ZX81 are your imagination and the number of hours in a day !

Std. Dev. 1K

This is an example of the kind of routine calculation work that the ZX81 really likes.

Engineers and statisticians often like to work out the mean and the standard deviation from that mean of a set of sample results. This program will do the calculations, on a set of up to about 20 numbers if you have only 1K RAM, or on a much larger set if you have more RAM fitted.

```
 10 LET B=0
 20 LET D=B
 30 PRINT AT 21,B;"NUMBER OF SAMPLES = ";
 40 INPUT N
 50 PRINT N
 60 DIM A(N)
100 FOR C=1 TO N
110 SCROLL
120 PRINT "SAMPLE ";C;" = ";
130 INPUT A(C)
140 PRINT A(C)
150 LET B=B+A(C)
160 NEXT C
170 SCROLL
180 LET B=B/N
190 PRINT "MEAN = ";B
200 FOR C=1 TO N
210 LET D=D+(A(C)-B)*(A(C)-B)
220 NEXT C
230 SCROLL
240 PRINT "STD DEV = ";SQR (D/N)
```

If you find the program useful, why not modify it to give a permanent record on the ZX printer ?

Ladder Analysis

Based on an Apple II program by S.J.Newett

This program illustrates how the ZX81 can be used for engineering applications which - only a few years ago - would have needed an expensive minicomputer system. If you are already using a mini or a time-sharing system for engineering calculations, try converting some of your favourite programs to run on the ZX81. I bet you'll be suprised at how well this 'toy' performs !

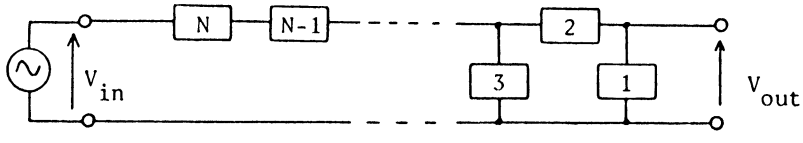
It calculates the gain (or loss), input impedance and output impedance over any range of frequencies for a passive ladder network of up to 10 series or parallel (shunt) branches.

Each parallel branch is a series R L C combination, but if any component is given the value zero then it is treated as a short-circuit. For example, if a parallel branch of the circuit you wanted to analyse was just an inductance, then you would enter the R and C values for that branch as zero. Similarly, series branches are treated as parallel R L C circuits, but any component given the value zero is assumed to be omitted.

Series and parallel branches may be intermixed freely, and are numbered in order starting at the output end of the network.

The program assumes that the driving source is a zero impedance voltage generator, and that the output of the network is connected to an infinite impedance voltage detector. The actual source and load impedances should be included within your model.

When you have entered the component values for all branches in your network, and the frequency range of interest, the program will display in succession the branch values, network gain, input impedance, and output impedance. Each screen full of results ends with a report 9 /.. , inviting you to enter CONT to view the next screen, COPY to get a print-out if you have a ZX printer connected, or STOP if you've had enough.



Generalised N-branch Ladder Network

Generalised 'Series' branch.

Generalised 'Parallel' branch.

Several 2-character variable names have been used in this program, but note that PI and LN are ZX81 functions, not variables.

```
100 REM *** LADDER ANALYSIS ***
110 FAST
120 GOSUB 9000
130 GOSUB 8000
140 IF I=1 THEN GOTO 230
150 GOSUB 2000
160 GOSUB 7000
170 GOSUB 6000
180 CLS
190 PRINT "MORE FREQUENCIES (Y/N) ?"
200 INPUT K$
210 IF K$="Y" THEN GOTO 160
220 CLS
230 STOP
240 RUN

1000 REM MULT T() BY QUADRATIC X
1010 FOR I=M(X)+1 TO 1 STEP -1
1020 IF X2<>0 THEN LET T(X,I+2)=T(X,I+2)+T(X,I)*X2
1030 IF X1<>0 THEN LET T(X,I+1)=T(X,I+1)+T(X,I)*X1
1040 LET T(X,I)=T(X,I)*X0
1050 NEXT I
1060 IF X2<>0 THEN LET M(X)=M(X)+2
1070 IF X2<>0 THEN RETURN
1080 IF X1<>0 THEN LET M(X)=M(X)+1
1090 RETURN

1200 REM MULT T() BY QUAD S, ADD RESULT TO T()
1210 FOR I=1 TO M(Y)+1
1220 LET T(X,I+S)=T(X,I+S)+T(Y,I)
1230 NEXT I
1240 IF M(Y)+S>M(X) THEN LET M(X)=M(Y)+S
1250 RETURN

1400 REM MAKE ANGLE -180 TO 180
1410 LET AN=AN-360*INT ((AN+180)/360)
1420 RETURN

2000 REM CALCULATE MATRICES
2010 FOR B=1 TO I-1
```

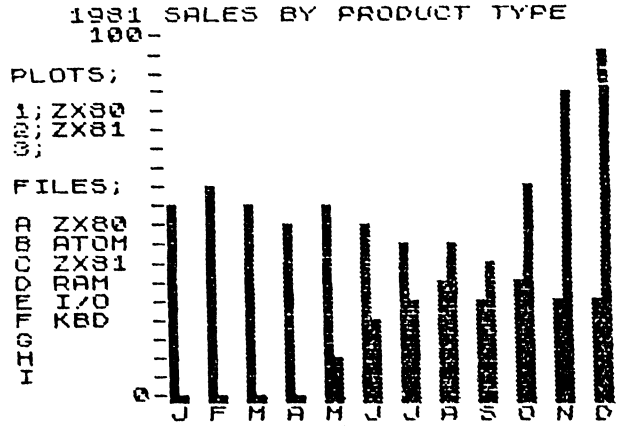
```
2020 LET X0=0
2030 LET X1=0
2040 LET X2=0
2050 IF E(B)=1 THEN GOTO 2100
2060 IF C(B)>0 THEN LET X2=C(B)
2070 IF R(B)>0 THEN LET X1=1/R(B)
2080 IF L(B)>0 THEN LET X0=1/L(B)
2090 GOTO 2130
2100 IF C(B)>0 THEN LET X0=1/C(B)
2110 IF R(B)>0 THEN LET X1=R(B)
2120 IF L(B)>0 THEN LET X2=L(B)
2130 LET S=1
2140 IF X0>0 THEN GOTO 2190
2150 LET X0=X1
2160 LET X1=X2
2170 LET X2=0
2180 LET S=0
2190 FOR C=1 TO 2
2200 LET X=C+2*E(B)
2210 LET Y=C+2*(1-E(B))
2220 GOSUB 1000
2230 GOSUB 1200
2240 LET X=Y
2250 GOSUB 1000
2260 NEXT C
2270 LET X=5
2280 GOSUB 1000
2290 NEXT B
2300 LET U=0
2310 FOR I=1 TO 5
2320 IF M(I)>U THEN LET U=M(I)
2330 NEXT I
2340 RETURN

3000 REM SAFE DIVISION
3010 IF ABS DD<1E-30 THEN LET DD=1E-30
3020 IF ABS DV<1E-30 THEN LET DV=1E-30
3030 IF 1E-30*ABS DD>ABS DV THEN GOTO 3060
3040 LET QU=DD/DV
3050 RETURN
3060 LET QU=1E30*SGN DD/SGN DV
3070 RETURN

6000 REM DISPLAY RESULTS
6010 FOR I=2 TO 6 STEP 2
6020 CLS
6030 PRINT "FREQ      ";
6040 IF I=2 THEN PRINT "GAIN (DB)";
6050 IF I=4 THEN PRINT "I/P OHMS ";
6060 IF I=6 THEN PRINT "O/P OHMS ";
6070 PRINT "      ANGLE"
6080 FOR L=1 TO 20
```

```
6Ø9Ø PRINT AT L+1,Ø;Z(L,1);TAB 9;Z(L,I);TAB 25;INT (1Ø*Z(L,I+1)
     +.5)/1Ø
61ØØ NEXT L
611Ø STOP
612Ø NEXT I
613Ø RETURN

7ØØØ REM FREQUENCY ANALYSIS
7Ø1Ø GOSUB 85ØØ
7Ø2Ø LET NF=1
7Ø3Ø LET F=F1+(NF-1)*F2
7Ø4Ø LET W=2*PI*F
7Ø5Ø LET V=1
7Ø6Ø FOR J=1 TO 5
7Ø7Ø LET K(J,1)=Ø
7Ø8Ø LET K(J,2)=Ø
7Ø9Ø NEXT J
71ØØ LET AI=-1
711Ø FOR I=Ø TO U
712Ø LET AI=-AI
713Ø LET BI=1+(AI<Ø)
714Ø FOR J=1 TO 5
715Ø LET K(J,BI)=K(J,BI)+V*T(J,I+1)
716Ø NEXT J
717Ø LET V=AI*V*W
718Ø NEXT I
719Ø FOR J=1 TO 5
72ØØ LET P(J,1)=SQR (K(J,1)*K(J,1)+K(J,2)*K(J,2))
721Ø LET DD=K(J,2)
722Ø LET DV=K(J,1)
723Ø GOSUB 3ØØØ
724Ø LET P(J,2)=(18Ø/PI)*ATN OU
725Ø IF K(J,1)<Ø THEN LET P(J,2)=P(J,2)+18Ø
726Ø NEXT J
74ØØ LET Z(NF,1)=F
741Ø LET DD=P(5,1)
742Ø LET DV=P(1,1)
743Ø GOSUB 3ØØØ
744Ø LET Z(NF,2)=2Ø*.43429*LN QU
745Ø LET AN=P(5,2)-P(1,2)
746Ø GOSUB 14ØØ
747Ø LET Z(NF,3)=AN
748Ø LET DD=P(1,1)
749Ø LET DV=P(3,1)
75ØØ GOSUB 3ØØØ
751Ø LET Z(NF,4)=QU
752Ø LET AN=P(1,2)-P(3,2)
753Ø GOSUB 14ØØ
754Ø LET Z(NF,5)=AN
755Ø LET DD=P(2,1)
756Ø LET DV=P(1,1)
757Ø GOSUB 3ØØØ
```

```
7580 LET Z(NF,6)=QU
7590 LET AN=P(2,2)-P(1,2)
7600 GOSUB 1400
7610 LET Z(NF,7)=AN
7620 IF NF=20 THEN RETURN
7630 LET NF=NF+1
7640 GOTO 7030

8000 REM INPUT BRANCHES
8010 FOR I=1 TO 10
8020 CLS
8030 PRINT "BRANCH ";I;"     TYPE"
8040 PRINT AT 2,5;"S - FOR SERIES BRANCH"
8050 PRINT TAB 5;"P - FOR PARALLEL"
8060 PRINT TAB 5;"E - TO END"
8070 INPUT K$
8080 IF K$="E" THEN GOTO 8300
8090 IF K$<>"S" AND K$<>"P" THEN GOTO 8070
8100 PRINT AT 0,18;K$
8110 LET E(I)=1 AND K$="P"
8120 PRINT AT 6,1;"INDUCTANCE (HENRIES) ="
8130 INPUT L(I)
8140 PRINT AT 6,25;L(I)
8150 PRINT AT 7,1;"RESISTANCE (OHMS)    ="
8160 INPUT R(I)
8170 PRINT AT 7,25;R(I)
8180 PRINT AT 8,1;"CAPACITANCE(FARADS)  ="
8190 INPUT C(I)
8200 NEXT I
8300 IF I=1 THEN RETURN
8310 CLS
8320 PRINT "BR    IND      RES     CAP"
8330 FOR L=1 TO I-1
8340 PRINT AT L+2,0;L;TAB 3;("S " AND E(L)=0);("P " AND E(L)<>0
     );L(L);TAB 14;R(L);TAB 23;C(L)
8350 NEXT L
8360 STOP
8370 RETURN

8500 REM GET FREQ RANGE
8510 CLS
8520 PRINT "LOWEST FREQ (HZ)  ?"
8530 INPUT F1
8540 PRINT AT 0,17;F1
8550 PRINT AT 2,0;"FREQ STEP (HZ)   ?"
8560 INPUT F2
8570 RETURN

9000 REM INITIALISE
9010 DIM L(10)
9020 DIM R(10)
9030 DIM C(10)
```

```
9040 DIM E(10)
9050 DIM T(5,20)
9060 DIM M(5)
9070 DIM K(5,2)
9080 DIM P(5,2)
9090 DIM Z(20,7)
9100 LET T(1,1)=1
9110 LET T(4,1)=1
9120 LET T(5,1)=1
9130 RETURN
```

G.P.G.P.



This General Purpose Graph Plotter can be used as a 'live' aid during presentations of business figures, or for the preparation of printed reports.

It can hold up to 9 files, each containing values for each of the 12 months of a year, and can plot up to 3 files on the screen in bar chart form at any time. A useful feature is that any of the three bar charts displayed can be replaced by another one on demand while the other two remain un-affected. Different sets of files can be saved on tape under different names.

Whenever the program is waiting for instructions it will display the familiar L cursor at the bottom left of the screen. You can then enter;

EM — to change the maximum value of the graphs. This changes the scaling factor so the graphs have to be re-plotted after using this command.

ET — to change the title shown at the top of the display.

C — to clear any plotted graphs from the screen.

S — to save the program with the current data files to tape, using the title shown at the top of the screen as the name.

EA to EI — to enter data into one of the nine data files A to I, or to change data already entered.

To plot a graph from one of the data files, enter the number 1, 2 or 3 according to whether you want the bars plotted in the left, middle or right position, followed by a letter A-I specifying which file you want plotted. For example, to plot the contents of file B on the left side, enter '1B'.

```
  1 REM G.P.G.P.
  2 GOTO 9000

100 REM BAR PLOT
110 FOR C=2 TO B
120 PLOT D,C
130 NEXT C
140 IF C>41 THEN RETURN
150 FOR C=C TO 41
160 UNPLOT D,C
170 NEXT C
180 RETURN

1000 REM MAIN ROUTINE
1010 INPUT A$
1020 IF A$="C" THEN GOTO 2000
1030 IF A$="S" THEN GOTO 3000
1040 IF LEN A$<>2 THEN GOTO 1010
1050 IF A$(1)="E" THEN GOTO 5000
1060 IF A$(1)>"0" AND A$(1)<"4" THEN GOTO 6000
1070 GOTO 1010

2000 REM CLEAR SCREEN
2010 FAST
2020 CLS
2030 PRINT T$;AT 1,0;M$;AT 3,0;"PLOTS;";AT 20,6;"0"
2040 FOR A=1 TO 20
2050 PRINT AT A,7;"-";Z$
2060 NEXT A
2070 PRINT TAB 8;L$
2080 FOR A=1 TO 3
```

```
2Ø9Ø PRINT AT 4+A,Ø;A;";"
21ØØ NEXT A
211Ø GOSUB 4ØØØ
212Ø SLOW
213Ø GOTO 1ØØØ

3ØØØ REM SAVE
3Ø1Ø LET B$=T$
3Ø2Ø IF B$(1)<>" " OR LEN B$=1 THEN GOTO 3Ø5Ø
3Ø3Ø LET B$=B$(2 TO )
3Ø4Ø GOTO 3Ø2Ø
3Ø5Ø IF B$(LEN B$)<>" " OR LEN B$=1 THEN GOTO 3Ø8Ø
3Ø6Ø LET B$=B$( TO LEN B$-1)
3Ø7Ø GOTO 3Ø5Ø
3Ø8Ø IF B$=" " THEN LET B$="GRAPH"
3Ø9Ø PRINT AT 21,Ø;"START RECORDER THEN PRESS KEY Y"
31ØØ IF INKEY$<>"Y" THEN GOTO 31ØØ
311Ø SAVE B$
312Ø PRINT AT 21,Ø;Z$,AT 21,8;L$
313Ø GOTO 1ØØØ

4ØØØ REM NAME LIST
4Ø1Ø PRINT AT 9,Ø;"FILES"
4Ø2Ø FOR A=1 TO 9
4Ø3Ø PRINT AT A+1Ø,Ø;CHR$ (37+A);" ";N$(A)
4Ø4Ø NEXT A
4Ø5Ø RETURN

45ØØ REM DELETE NAME LIST
451Ø FOR A=9 TO 19
452Ø PRINT AT A,Ø;S$
453Ø NEXT A
454Ø RETURN

5ØØØ REM EDIT
5Ø1Ø IF A$(2)<"A" OR (A$(2)>"I" AND A$(2)<>"M" AND
     A$(2)<>"T") THEN GOTO 1ØØØ
5Ø2Ø GOSUB 45ØØ
5Ø4Ø IF A$(2)="M" THEN GOTO 53ØØ
5Ø5Ø IF A$(2)="T" THEN GOTO 54ØØ
51ØØ PRINT AT 12,Ø;"GRAPH";A$(2)
511Ø LET G=CODE A$(2)-37
512Ø PRINT AT 14,Ø;"NAME ?"
513Ø INPUT N$(G)
514Ø PRINT AT 14,Ø;N$(G),AT 16,Ø;"DATA"
515Ø PRINT "FOR";AT 18,Ø;"MONTH:"
516Ø FOR A=1 TO 12
517Ø PRINT AT 19,Ø;A;" ?"
518Ø INPUT D(G,A)
519Ø NEXT A
52ØØ GOSUB 4ØØØ
521Ø GOTO 1ØØØ

5300 PRINT AT 18,Ø;"NEW MAX";AT 19,Ø;"VALUE ?"
531Ø INPUT M
```

```
5320 LET B$=STR$ M
5330 LET M$=Z$
5340 LET M$(8-LEN B$ TO )=B$
5350 GOTO 2000

5400 PRINT AT 18,0;"NEW";AT 19,0;"TITLE ?"
5410 INPUT B$
5420 LET T$=Z$
5430 IF LEN B$>32 THEN LET B$=B$( TO 32)
5440 LET T$((33-LEN B$)/2 TO )=B$
5450 PRINT AT 0,0;T$
5460 GOTO 5200

6000 REM PLOT
6010 IF A$(2)<"A" OR A$(2)>"I" THEN GOTO 1000
6020 LET G=CODE A$(2)-37
6030 LET P=CODE A$(1)-CODE "0"
6040 PRINT AT 4+P,2;N$(G)
6050 FOR M=1 TO 12
6060 LET B=2+INT (D(G,M)*40/VAL M$+.5)
6070 IF B>41 THEN LET B=41
6080 LET D=P+11+4*M
6090 GOSUB 100
6100 NEXT M
6110 GOTO 1000

9000 REM INITIALISE
9010 DIM T$(32)
9020 DIM M$(7)
9030 DIM N$(9,5)
9040 DIM D(9,12)
9050 DIM Z$(24)
9060 DIM S$(7)
9070 LET M$(7)="1"
9080 LET L$="J F M A M J J A S O N D"
9090 GOTO 2000
```

Items that may be worthy of note in the program are the routines to centralise the title on the screen (5420- 5450), to right justify the max value (5320-5340) and to strip leading and trailing spaces from T$ so that it can be used for the name when saving the program and data to tape (3010-3070). The heart of the plotting routine is placed at the beginning of the program listing (lines 100-180) for speed.

If your year starts in a month other than January, change L$ (line 9080) accordingly.

Bank

```
                    STATEMENT

1/1/82
 I.B.M.                          £1000.00CR
1/1/82
 SINCLAIR                        £2000.00CR
2/1/82      CHQ/11111
 ME                               £500.00
5/1/82      CHQ/11112
 ELECTRICITY                       £38.50
10/1/82     CHQ/11113
 CLIVE                            £500.00
19/1/82     CHQ/11114
 TELEPHONE                         £25.73


                    BALANCE:    £1935.77

LAST ENTRY:6
ENTER NUMBER TO REVIEW,
OR NEWLINE FOR MENU
```

By J.Durst

This program will keep track of your personal bank account. It holds a file of up to 100 entries, with name, date and amount, and will display a statement of any 7 consecutive entries together with the current balance. Credit entries (payments into the bank account) are marked, as are debit (overdraft) balances.

After entering data, use the 'SV' option to save an updated file to tape. When the file is full, the first 90 entries can be deleted, and the last 10 moved up, to start a new ledger. Dates should be entered as 4 digits e.g. 7th. July as 0707. The year can be changed by altering A$ (line 150).

```
 10 LET I=0
 20 LET RESET=0
 30 DIM D$(100,4)
 40 DIM C(100)
 50 DIM N$(100,15)
 60 DIM S(100)

100 LET MENU=380
110 LET DEBIT=530
120 LET CREDIT=830
130 LET STATEMENT=1090
140 LET TOTAL=0
150 LET A$="/82"

200 LET K=0
210 GOTO MENU
220 REM SUBROUTINE PRINT DATE
230 IF D$(J,1)="0" THEN GOTO 250
240 PRINT D$(J,1);
250 PRINT D$(J,2);"/";
260 IF D$(J,3)="0" THEN GOTO 280
270 PRINT D$(J,3);
280 PRINT D$(J,4);A$;
290 RETURN
```

```
300 REM SUBROUTINE POUNDS/PENCE
310 PRINT "£";INT ABS S(J);".";
320 LET PENCE=INT (100*ABS S(J)-100*INT ABS S(J)+.5)
330 IF PENCE=0 THEN PRINT "00";
340 IF PENCE=0 THEN RETURN
350 IF PENCE<10 THEN PRINT "0";
360 PRINT PENCE;
370 RETURN

380 REM MENU
390 PRINT AT 3,9;"BANK BALANCE"
400 PRINT AT 5,8;"(ACCOUNT NAME)"
410 PRINT AT 6,13;A$
420 PRINT AT 9,4;"CR - FOR CREDIT ENTRY";AT 11,4;"DB - FOR
    DEBIT ENTRY"
430 IF I<>0 THEN PRINT AT 13,4;"ST - FOR STATEMENT"
440 PRINT AT 15,4;"SV - TO SAVE ON TAPE"
450 IF I=99 THEN PRINT AT 16,0;"*LAST ENTRY BEFORE FILE FULL*"
460 INPUT Y$
470 CLS
480 IF Y$="CR" THEN GOTO CREDIT
490 IF Y$="DB" THEN GOTO DEBIT
500 IF Y$="ST" THEN GOTO STATEMENT
510 IF Y$="SV" THEN SAVE "BANK"
520 GOTO 380

530 REM DEBIT ENTRY
540 LET I=I+1
550 LET J=I
560 PRINT AT 3,10;"DEBIT ENTRY"

570 REM ENTER DATE
580 PRINT AT 5,4;"DATE: ";
590 INPUT D$(J)
600 GOSUB 220
610 PRINT
620 REM ENTER CHEQUE NO.
630 PRINT AT 7,4;"CHEQUE NUMBER: ";
640 INPUT C(J)
650 PRINT "/";C(J)
660 REM ENTER PAYEE NAME
670 PRINT AT 9,4;"PAYEES NAME: ";
680 INPUT N$(J)
690 PRINT N$(J)
700 REM ENTER AMOUNT
710 PRINT AT 11,4;"AMOUNT: ";
720 INPUT S(J)
730 LET S(J)=-S(J)
740 GOSUB 300
750 PRINT AT 20,0;"NEWLINE - FOR MENU"
760 PRINT AT 21,4;"COR - FOR CORRECTION"
770 INPUT Y$
780 CLS
```

```
 79Ø IF Y$="COR" THEN GOTO 56Ø
 8ØØ IF I=1ØØ THEN GOTO STATEMENT
 81Ø GOTO MENU
 83Ø REM CREDIT ENTRY
 84Ø LET I=I+1
 85Ø LET J=I
 86Ø PRINT AT 3,9;"CREDIT ENTRY"
 87Ø REM ENTER DATE
 88Ø PRINT AT 5,4;"DATE: ";
 89Ø INPUT D$(J)
 9ØØ GOSUB 22Ø
 91Ø PRINT
 92Ø REM ENTER NAME
 93Ø PRINT AT 7,4;"PAYERS NAME: ";
 94Ø INPUT N$(J)
 95Ø PRINT N$(J)
 96Ø REM ENTER AMOUNT
 97Ø PRINT AT 9,4;"AMOUNT: ";
 98Ø INPUT S(J)
 99Ø GOSUB 3ØØ
1ØØØ PRINT AT 2Ø,Ø;"NEWLINE - FOR MENU"
1Ø1Ø PRINT AT 21,4;"COR - TO CORRECT ENTRY"
1Ø2Ø INPUT Y$
1Ø3Ø CLS
1Ø4Ø IF Y$="COR" THEN GOTO 86Ø
1Ø5Ø IF I=1ØØ THEN GOTO STATEMENT
1Ø6Ø GOTO MENU
1Ø9Ø REM STATEMENT
11ØØ FAST
111Ø CLS
112Ø PRINT AT Ø,1Ø;"STATEMENT"
113Ø PRINT
114Ø FOR K=1ØØ TO 1 STEP -1
115Ø IF S(K)=Ø THEN NEXT K
116Ø LET Q=K
117Ø FOR J=Q-6 TO Q
118Ø IF J<1 THEN NEXT J
12ØØ GOSUB 22Ø
121Ø IF C(J)=Ø THEN PRINT
122Ø IF C(J)=Ø THEN GOTO 124Ø
123Ø PRINT TAB 1Ø;"CHQ/";C(J)
124Ø PRINT " ";N$(J);TAB (24-LEN STR$ INT ABS S(J));
125Ø GOSUB 3ØØ
126Ø IF S(J)>Ø THEN PRINT "CR";
127Ø PRINT
13ØØ NEXT J
131Ø REM GET TOTAL
132Ø LET TOTAL=RESET
133Ø FOR T=1 TO K
134Ø LET TOTAL=TOTAL+S(T)
135Ø NEXT T
```

```
136Ø LET PENCE=INT (1ØØ*(ABS TOTAL-INT ABS TOTAL)+.5)
137Ø LET T$=STR$ ABS INT TOTAL
138Ø PRINT AT 17,11;"BALANCE: £";T$;".";
139Ø GOSUB 33Ø
14ØØ IF TOTAL<Ø THEN PRINT "*OD"
141Ø PRINT AT 19,Ø;"LAST ENTRY:";Q
142Ø IF I=99 THEN PRINT AT 19,Ø;"FINAL ENTRY BEFORE FILE
     FULL"
143Ø PRINT AT 2Ø,Ø;"ENTER NUMBER TO REVIEW,"
144Ø PRINT AT 21,Ø;"OR NEWLINE FOR MENU"
145Ø IF I=1ØØ THEN PRINT AT 21,Ø;"FILE FULL; NEWLINE TO
     RE-NUMBER"
146Ø SLOW
147Ø INPUT Y$
148Ø CLS
149Ø IF Y$="" AND I=1ØØ THEN GOTO 154Ø

15ØØ IF CODE Y$<29 OR CODE Y$>37 THEN GOTO MENU
151Ø IF VAL Y$>=1 AND VAL Y$<1Ø1 THEN LET Q=VAL Y$
152Ø IF VAL Y$>=1 AND VAL Y$<1Ø1 THEN GOTO 117Ø
153Ø CLS
1535 GOTO MENU

154Ø PRINT "FILE IS NOW FULL."
155Ø PRINT AT 4,Ø;"ST - FOR CURRENT STATEMENT"
156Ø PRINT AT 6,Ø;"NEWLINE WILL ERASE ALL BUT LAST 1Ø
     ENTRIES"
157Ø INPUT Y$
158Ø IF Y$<>"" THEN GOTO MENU
159Ø FAST
16ØØ LET T2=Ø
161Ø FOR J=91 TO 1ØØ
162Ø LET D$(J-9Ø,4)=D$(J,4)
163Ø LET C(J-9Ø)=C(J)
164Ø LET N$(J-9Ø,15)=N$(J,15)
165Ø LET S(J-9Ø)=S(J)
166Ø LET T2=T2+S(J)
167Ø NEXT J
168Ø LET RESET=TOTAL-T2
169Ø FOR J=11 TO 1ØØ
17ØØ LET D$(J,4)=""
171Ø LET C(J)=Ø
172Ø LET N$(J,15)=""
173Ø LET S(J)=Ø
174Ø NEXT J
175Ø LET I=1Ø
176Ø CLS
177Ø SLOW
178Ø GOTO STATEMENT
```

Linear Programming

By Lee Gentry

Linear Programming is a technique used widely in industry to plan for the best use of limited resources.

As a (rather parochial) example, let us suppose that you are to organise a film show at the local Church hall. The hall has 300 square metres of floor space, and you have 40 chairs. If you are going to charge £1.00 to sit, but only 40p to stand, and if a seated person takes up 4 square metres of floor while a standing person takes only half that space, then how many chairs should you put out if you want to take the most money at the door ? (For the sake of simplicity we will assume that the film is so good that there will be no empty spaces left.)

The problem can be described by two 'inequality' expressions;

```
1 x X + Ø x Y<=4Ø    (Max 40 chairs)
4 x X + 2 x Y<=3ØØ   (Max 300 square metres)
```

where X is the number of people sitting, and Y the number standing.

The objective is to maximise the 'take';

```
1 x X + .4 x Y
```

The program here will accept any set of linear inequalities of 2 variables (such as those shown above) and calculates the optimum values of the variables to maximise or minimise a given objective.

When run, it will first ask how many inequalities there are,then it will ask you to enter tham one at a time. They must be entered as a string of the form;

	number	*variable*	*number*	*variable*	*comparison*	*number*
e.g;	-3.5	* X	+ 7	* Y	> =	.5

The program treats '>' as '>=', and '<' as '<='. It will also recognise as a number an (optional) - sign followed by an integer, decimal number or simple fraction, thus

```
-3      5/7     2.7  are all valid 'numbers'
```

Any non-numeric expression entered is treated as being variable X or Y depending on its position in the input string. Thus;

```
                1 *SIT + Ø *STAND   <= 4Ø
is treated as   1 * X  + Ø * Y      <= 4Ø
```

The 'objective' is entered similarly, as a string

	number	*variable*	*number*	*variable*
e.g;	27	*X	-3.3	*Y
or;	1	*SIT	+.4	*STAND

```
  1 SLOW
  5 GOTO 900
 10 REM SOLVE AX+BY=C AND DX+EY=F
 20 FOR N=I TO O
 30 LET H=U+V*(N-I)
 40 FOR J=I TO T
 50 IF Z(H,I)=Q THEN GOTO 150
 60 LET A(N,J)=Z(H,J)/Z(H,I)
 70 NEXT J
 80 NEXT N
 90 IF A(I,I)=A(O,I) AND A(I,O)=A(O,O) THEN GOTO 220
100 IF Y(G)=Q THEN LET Y(G)=(A(I,T)-A(O,T))/(A(I,O)-A(O,O))
110 IF Y(G)=VAL C$ THEN LET Y(G)=A(R,T)
120 IF A(I,I)<>Q THEN LET X(G)=A(I,T)-A(I,O)*Y(G)
130 IF A(O,I)<> Q THEN LET X(G)=A(O,T)-A(O,O)*Y(G)
140 RETURN

150 LET A(N,I)=Q
160 LET A(N,O)=I
170 LET A(N,T)=Z(H,T)/Z(H,O)
180 LET R=N
190 LET J=3
200 LET Y(G)=VAL C$
210 GOTO 70

220 LET X(G)=VAL C$
230 LET Y(G)=VAL C$
240 RETURN

260 REM ALLOWED ?
270 FOR G=I TO K
280 IF X(G)=VAL C$ AND Y(G)=VAL C$ THEN GOTO 370
290 FOR S=I TO M
300 IF Z(S,4)>.5 THEN IF Z(S,I)*X(G)+Z(S,O)*Y(G)<Z(S,T)-1E-5
    THEN GOTO 330
310 IF Z(S,4)<-.5 THEN IF Z(S,I)*X(G)+Z(S,O)*Y(G)>Z(S,T)+1E-5
    THEN GOTO 330
320 GOTO 360

330 LET X(G)=VAL C$
340 LET Y(G)=VAL C$
350 LET S=K
360 NEXT S
370 NEXT G

400 REM OBJECTIVE
410 PRINT AT 17,0;Q$;Q$;Q$;Q$;Q$
420 SLOW
430 PRINT AT 17,0;"OBJECTIVE : ";
440 INPUT F$
450 PRINT F$
460 FAST
470 LET R=0
480 LET R=R+1
490 IF F$(R)=" " THEN GOTO 480
```

```
500 LET G=CODE F$(R)
510 IF (G>=27 AND G<38) OR G=22 THEN GOTO 550
520 IF R<LEN F$ THEN GOTO 480
530 SLOW
540 GOTO 640

550 LET W=R
560 LET R=R+1
570 IF R>LEN F$ THEN GOTO 600
580 LET G=CODE F$(R)
590 IF (G>=27 AND G<38) OR G=0 OR G=22 OR G=24 THEN GOTO 560
600 LET J(JJ)=VAL F$(W TO R-1)
610 LET JJ=JJ+1
620 LET R=R-1
630 GOTO 520

640 LET W=0
650 PRINT AT 20,0;"MIN OR MAX VALUE ?";TAB 0;
660 INPUT D$
670 PRINT D$
680 FAST
690 FOR G=I TO K
700 IF X(G)=VAL C$ AND Y(G)=VAL C$ THEN GOTO 790
710 LET Z=J(1)*X(G)+J(2)*Y(G)
720 IF W<>0 THEN GOTO 750
730 IF D$="MAX" THEN LET F=-999
740 IF D$="MIN" THEN LET F=VAL C$
750 IF D$="MIN" AND Z<=F THEN LET W=G
760 IF D$="MAX" AND Z>=F THEN LET W=G
770 IF D$="MAX" AND Z>=F THEN LET F=Z
780 IF D$="MIN" AND Z<=F THEN LET F=Z
790 NEXT G

800 PRINT AT 20,0;Q$;Q$
810 PRINT AT 20,0;D$;" IS ";J(1)*X(W)+J(2)*Y(W)
820 PRINT AT 21,0;"WHEN X = ";X(W);" Y = ";Y(W)
850 SLOW
860 STOP

900 REM INITIALISE
910 DIM Q$(32)
920 PRINT "HOW MANY INEQUALITIES"
930 INPUT M
940 PRINT AT 0,0;Q$
950 PRINT AT 0,0;M;" INEQUALITIES"
960 DIM A$(M,15)
970 DIM Z(M,4)
980 DIM J(2)
990 LET JJ=1
1000 LET I=1
1010 LET Q=0
1020 LET K=0
1030 LET O=2
1040 LET T=3
```

```
105Ø DIM A(0,T)
106Ø FOR G=I TO M-I
107Ø LET K=K+G
108Ø NEXT G
109Ø IF K<1 THEN LET K=1
1100 LET C$="999"
1110 LET W=Q
1120 LET Z=W
1130 DIM Y(K)
1140 DIM X(K)

1160 REM INPUT
1170 FOR S=1 TO M
1180 LET J=I
1190 LET R=Ø
1200 PRINT AT 2Ø,Ø;"INEQUALITY ";S
1210 INPUT B$
1220 FAST
1230 LET R=R+1
1240 IF B$(R)=" " THEN GOTO 1290
1250 LET G=CODE B$(R)
1260 IF (G>=27 AND G<38) OR G=22 THEN GOTO 1330
1270 IF G=18 OR G=220 THEN LET Z(S,4)=I
1280 IF G=19 OR G=219 THEN LET Z(S,4)=-I
1290 IF R<LEN B$ THEN GOTO 1230
1300 SLOW
1310 NEXT S
1320 GOTO 1500

1330 LET W=R
1340 LET R=R+1
1350 IF R>LEN B$ THEN GOTO 1380
1360 LET G=CODE B$(R)
1370 IF (G>=27 AND G<38) OR G=Ø OR G=22 OR G=24 THEN GOTO 1340
1380 LET Z(S,J)=VAL B$(W TO R-1)
1390 LET J=J+1
1400 PRINT AT S*2,Ø;B$
1410 LET R=R-1
1420 GOTO 1280

1500 REM MAIN PART
1510 FAST                    4 INEQUALITIES
1520 LET W=Ø
1530 LET G=I                 1 SIT + 0 STAND >= 0
1540 FOR U=I TO M            0 SIT + 1 STAND >= 0
1550 FOR V=I TO M-U          1 SIT + 0 STAND <= 40
1560 GOSUB 10                4 SIT + 2 STAND <= 300
1570 LET G=G+1
1580 NEXT V
1590 NEXT U
1600  GOTO 250

OBJECTIVE : 1.00 SIT + .40 STAND

MAX IS 68
WHEN X = 40 Y = 70
```

Simultaneous Equations

Lee Gentry

This program will solve any set of simultaneous linear equations (N equations of N unknowns), provided that the set is not interdependent or inconsistent.

```
  5 REM SIMULTANEOUS EQUATIONS
 10 DIM Q$(32)
 15 LET J=1
 20 LET L=0
 30 LET I=0
 40 PRINT AT 21,0;"HOW MANY VARIABLES DO YOU HAVE ?"
 50 INPUT N
 60 PRINT AT 21,0;Q$ 1*X1+ 1*X2+ 1*X3+ 1*X4+   = 10
 70 DIM Y(N)         1*X1+ 2*X2+ 3*X3+ 4*X4+   = 20
 80 DIM X(N)
 90 DIM D(N+1,N+1)   2*X1+ 1*X2+ 1*X3+ 3*X4+   = 15
100 DIM C(N+1,N+1)   1*X1+ 1*X2+ 3*X3+ 5*X4+   = 15
110 FOR A=1 TO N
120 PRINT AT 0,18;N;" EQUATIONS"
130 PRINT AT 0,0;"EQUATION ";A
140 PRINT AT 18,0;"FOR EACH EQUATION TYPE EACH COEFFICIENT
    (INCLUDING THE CONSTANT ON RHS OF THE EQUATION),THEN
    NEWLINE"
150 FOR B=1 TO N+1
160 PRINT AT B,0;" X";B;                          X1 = 4
170 INPUT D                                       X2 = 3
180 LET C(A,B)=D                                  X3 = -3
190 LET D(A,B)=D                                  X4 = 1
200 PRINT " : ";C(A,B)
210 NEXT B
220 PRINT AT 18,0;Q$;" FOR CORRECTION:TYPE C,NEWLINE,NEW
    COEFFICIENT"
230 PRINT AT 21,0;"TYPE NEWLINE TO CONTINUE"
240 INPUT E$
250 CLS
260 IF E$="C" THEN GOTO 120
270 CLS
280 NEXT A

300 REM PIVOT AND CALCULATION
310 FOR A=1 TO N
320 LET P=C(A,A)
330 FOR B=A+1 TO N
340 IF ABS P>= ABS C(B,A) THEN GOTO 410
350 LET P=C(B,A)

360 FOR D=1 TO N+1
370 LET F=C(A,D)
380 LET C(A,D)=C(B,D)
```

```
390 LET C(B,D)=F
400 NEXT D
410 NEXT B
420 FOR D=N+1 TO A STEP -1
430 IF C(A,A)=0 THEN GOTO 800
440 LET C(A,D)=C(A,D)/C(A,A)
450 NEXT D
460 FOR B=A+1 TO N
470 FOR D=N+1 TO A STEP -1
480 LET C(B,D)=C(B,D)-C(A,D)*C(B,A)
490 NEXT D
500 NEXT B
510 NEXT A

550 REM CALCULATE
560 LET X(N)=C(N,N+1)
570 FOR A=N-1 TO 1 STEP -1
580 LET X(A)=C(A,N+1)
590 FOR D=N TO A+1 STEP -1
600 LET X(A)=X(A)-C(A,D)*X(D)
610 NEXT D
620 NEXT A

650 REM OUTPUT
660 SLOW
670 FOR A=1 TO N
680 FOR B=1 TO N-1
690 PRINT D(A,B);"*X";B;"+ ";
700 NEXT B
710 PRINT D(A,N);"*X";N;" = ";D(A,N+1);TAB 0
720 NEXT A
730 IF L=1 THEN RETURN
740 FOR A=1 TO N
750 PRINT AT 20-N+A,0;"X";A;" =";TAB 4+(1 AND X(A)>=0);X(A)
760 NEXT A
770 GOTO 1000

800 REM DEPENDENT OR INCONSISTENT
810 FOR K=A TO N
820 FOR H=A TO N
830 LET C(K,H)=C(K,H+1)
840 NEXT H
850 LET C(K,N)=0
860 NEXT K
870 IF I=A THEN GOTO 910
880 LET I=A
890 LET J=I
900 GOTO 320
910 LET J=J+1
920 IF J<N-A+1 THEN GOTO 320
930 LET L=1
940 GOSUB 660
950 FOR K=1 TO N
```

```
 96Ø IF C(K,N+1)<>Ø THEN PRINT AT 21,Ø;"INCONSISTENT"
 97Ø IF C(K,N+1)<>Ø THEN GOTO 1ØØØ
 98Ø NEXT K
 99Ø PRINT AT 21,Ø;DEPENDENT"
1ØØØ STOP
```

Members

This program should make life easier for hard pressed club secretaries. It holds details for up to 115 members on tape, and lets you alter or print it out in a variety of ways.

The data held for each member is;

- The membership number. These are allocated sequentially to new members.
- Name, telephone number and address ; a maximum of 94 characters is allocated for these three items.
- Two 3-character 'keys', which can be used to hold information such as the class of membership, and the month in which the member's subscription is due.

If your club is a very large one, then a separate cassette can be used to hold information for each block of 100 members. Or, if you have 32K or 48K of RAM fitted to your ZX81, then you can make the program handle 278 or 441 members by altering line 100.

When the program is first run, it will ask you for the first membership number. After this, it will automatically allocate new numbers to new members.

Once you have entered some data, then if the program should ever stop for any reason, don't use RUN to re-start it as this will destroy all of the records. Use GOTO 1ØØØ instead.

Once you have entered the first member's number, the screen will look like;

```
NUMB :
NAME :
 TEL :
ADDR :
     :
     :
     :
     :
KEY 1 :
KEY 2 :
NUMBER  +  -  C  N  P  S ■
```

The black square is a 'cursor' used throughout the program to draw your attention to the next input required. In this case it is against the main menu line, and you have a choice of entering;

NUMBER ; entering a valid membership number will display the details for that member.

\+ ; will display the details for the next highest numbered member.

\- ; will display the details for the previous member.

C ; allows you to change the member's details currently being displayed.

N ; to add a new member.

P ; to print out details of members.

S ; to save the program with its data on tape.

If you enter C or N , then the bottom 'menu' line will vanish, and the black cursor will move to just after 'NAME :'.
You should then enter the member's name. On pressing the NEWLINE key, the cursor will move down to prompt you to enter the telephone number. Continue entering lines in this way until, after you have entered the 'KEY 2' data, the main menu will re-appear. If you enter too many characters against any item, the program will ignore the excess, and will only show on the screen those characters that it has enough room to store.

If you choose the P (Print) option, then the display will change to;

```
KEY 1 : ■
KEY 2 :
 FULL :
```

and wait for you to enter data against these three lines before starting to print. For the KEY 1 and KEY 2 lines, you have a choice of entering nothing (by simply pressing NEWLINE), or a 3-character key. If you enter a key, then only those members' records which have a matching key will be printed. This lets you - say - print out a list of those members whose subscriptions were due in 'AUG'.
The third line (FULL) requires a response of Y or N. Entering Y will cause all of the details to be printed out for each member, so that you could use the printer output as address labels. Entering N prints out only the membership number, keys and the member's name. When the printout has finished, the program will return to the main menu.

```
100 LET M=115
110 DIM D$(M,100)
120 DIM X$(3)
130 DIM Y$(25)
140 DIM Z$(32)
```

```
200 CLS
210 PRINT AT 19,0;"NEW DATA FILE"
220 PRINT AT 21,0;"FIRST MEMBERS NUMBER ? ■"
230 GOSUB 9000
240 IF I=0 THEN GOTO 220
250 LET F=I
260 LET N=I
270 LET C=0

1000 CLS
1010 PRINT AT 3,1;"NUMB :";AT 5,1;"NAME :"
1020 PRINT AT 7,2;"TEL :";AT 9,1;"ADDR :"
1030 PRINT AT 10,6;":";AT 11,6;":";AT 12,6;":";AT 13,6;":"
1040 PRINT AT 15,0;"KEY 1 :";AT 16,0;"KEY 2 :"

2000 PRINT AT 21,0;"NUMBER  -  C  N  P  S ■"
2010 GOSUB 9000
2020 PRINT AT 21,0;Z$
2030 IF I>0 THEN GOTO 3000
2040 IF I$="+" THEN GOTO 3100
2050 IF I$="-" THEN GOTO 3200
2060 IF I$="C" THEN GOTO 4000
2070 IF I$="N" THEN GOTO 4100
2080 IF I$="P" THEN GOTO 5000
2090 IF I$="S" THEN GOTO 6000
2100 GOTO 2000

3000 IF I<F OR I>=N THEN GOTO 2000
3010 LET C=I
3020 GOTO 3300
3100 IF C>=N-1 THEN GOTO 2000
3110 LET C=C+1
3120 GOTO 3300
3200 IF C<=F THEN GOTO 2000
3210 LET C=C-1
3300 FOR A=3 TO 16
3310 PRINT AT A,7;Y$
3320 NEXT A
3330 PRINT AT 3,8;C
3340 LET P=7
3350 FOR L=5 TO 13
3360 IF L=6 OR L=8 THEN LET L=L+1
3370 PRINT AT L,8;
3380 IF P>100 THEN GOTO 3500
3390 LET Q=CODE D$(C+1-F,P)

3400 LET P=P+1
3410 IF Q>127 THEN GOTO 3440
3420 PRINT CHR$ Q;
3430 GOTO 3380
3440 PRINT CHR$ (Q-128);
3450 NEXT L
3500 PRINT AT 15,8;D$(C+1-F, TO 3)
```

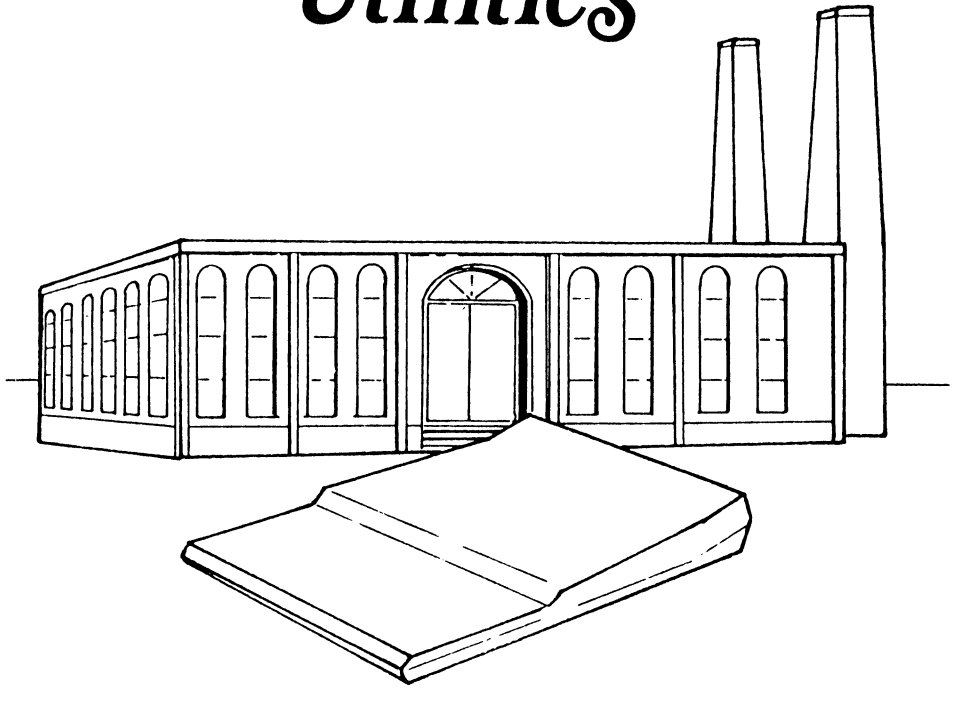
```
351Ø PRINT AT 16,8;D$(C+1-F,4 TO 6)
352Ø GOTO 2ØØØ

4ØØØ IF C<F OR C>=N THEN GOTO 2ØØØ
4Ø1Ø GOTO 42ØØ
41ØØ IF N< F+M THEN GOTO 413Ø
411Ø PRINT AT 19,Ø;"FILE FULL"
412Ø GOTO 2ØØØ
413Ø LET C=N
414Ø LET N=N+1
415Ø FOR A=3 TO 16
416Ø PRINT AT A,7;Y$
417Ø NEXT A

42ØØ PRINT AT 21,Ø;Z$;AT 3,8;C
421Ø LET P=7
422Ø FOR L=5 TO 13
423Ø IF L=6 OR L=8 THEN LET L=L+1
424Ø PRINT AT L,7;"■"
425Ø LET J$=Y$
426Ø IF P>1ØØ THEN GOTO 437Ø
427Ø INPUT I$
428Ø IF LEN I$>24 THEN LET I$=I$( TO 24)
429Ø IF I$="" THEN LET I$=" "
43ØØ LET K=LEN I$
431Ø IF P+K>1ØØ THEN LET I$=I$( TO 1Ø1-P)
432Ø LET K=LEN I$
433Ø LET J$(2 TO K+1)=I$
434Ø LET I$(K)=CHR$ (CODE I$(K)+128)
435Ø LET D$(C+1-F,P TO P+K-1)=I$
436Ø LET P=P+K
437Ø PRINT AT L,7;J$
438Ø NEXT L

44ØØ PRINT AT 15,7;"■"
441Ø INPUT I$
442Ø LET D$(C+1-F, TO 3)=I$
443Ø PRINT AT 15,7;" ";D$(C+1-F, TO 3);AT 16,7;"■"
444Ø INPUT I$
445Ø LET D$(C+1-F,4 TO 6)=I$
446Ø PRINT AT 16,7;" ";D$(C+1-F,4 TO 6)
447Ø GOTO 2ØØØ

5ØØØ CLS
5Ø1Ø PRINT AT 5,Ø;"KEY 1 :";AT 7,Ø;"KEY 2 :";AT 9,1;"FULL :"
5Ø2Ø PRINT AT 5,7;"■"
5Ø3Ø INPUT I$
5Ø4Ø LET X$=I$
5Ø5Ø IF I$<>"" THEN LET I$=X$
5Ø6Ø PRINT AT 5,7;" ";I$;AT 7,7;"■"
5Ø7Ø INPUT J$
5Ø8Ø LET X$=J$
5Ø9Ø IF J$<>"" THEN LET J$=X$
51ØØ PRINT AT 7,7;" ";J$;AT 9,7;"■"
```

```
5110 INPUT K$
5120 IF K$="" THEN GOTO 5100
5130 IF LEN K$>1 THEN LET K$=K$( TO 1)
5140 IF K$<>"Y" AND K$<>"N" THEN GOTO 5100
5150 PRINT AT 9,7;" ";K$
5200 LPRINT
5210 FOR C=F TO N-1
5220 LET L$=D$(C-F+1)
5230 IF I$<>"" AND I$<>L$( TO 3) THEN GOTO 5390
5240 IF J$<>"" AND J$<>L$(4 TO 6) THEN GOTO 5390
5250 IF K$="Y" THEN LPRINT
5260 LPRINT C;L$( TO 6);" ";
5270 LET P=7
5280 FOR L=1 TO 7
5290 IF K$="N" AND L>1 THEN GOTO 5380
5300 IF P>100 THEN GOTO 5370
5310 LET Q=CODE L$(P)
5320 LET P=P+1
5330 IF Q>127 THEN GOTO 5360
5340 LPRINT CHR$ Q;
5350 GOTO 5310
5360 LPRINT CHR$ (Q-128);
5370 LPRINT
5380 NEXT L
5390 NEXT C
5400 LPRINT
5410 GOTO 1000

6000 SAVE "MEMBERS"
6010 GOTO 1000

9000 INPUT I$
9010 LET I=0
9020 IF I$="" THEN RETURN
9030 FOR A=1 TO LEN I$
9040 IF (I$(A)<"0" OR I$(A)>"9") AND I$(A)<>" " THEN GOTO 9060
9050 NEXT A
9060 IF A>1 THEN LET I=VAL I$( TO A-1)
9070 RETURN
```

```
NUMB  :  100
NAME  :  TIMEDATA LTD.
 TEL  :  411125
ADDR  :  57 SWALLOWDALE
      :  BASILDON
      :  ESSEX
      :  SS16 5UG
      :
KEY 1 :
KEY 2 :  K2

NUMBER  +  -  D  M  P  S  ■
```

Chapter 6

Utilities



Decimal Peeker

The following simple program will list out the contents of any 22 consecutive memory locations (ROM or RAM). Enter the address you want the list to start at, and the program will display the 22 addresses with the contents of these locations in decimal.

When the screen is full the program will stop with the report 5/4Ø , allowing you to enter COPY to get a permanent record on the ZX printer, CONT to display the next 22 locations, or quit.

```
1Ø PRINT "ADDR ?"
2Ø INPUT A
3Ø CLS
4Ø PRINT A;"  ";PEEK A
5Ø LET A=A+1
6Ø GOTO 4Ø
```

The program is very useful for examining how programs are stored by the ZX81 in RAM. If you add a test line, say;

```
5 LET A=5
```

and then RUN the program, giving it the starting address 165Ø9 (the start of the ZX81's program storage area),you will get;

```
165Ø9   Ø  ┐
1651Ø   5  ┘  line number 5  (Ø * 256 + 5)
16511   11 ┐
16512   Ø  ┘  number of bytes following in this line
                (11 + Ø * 256)
16513   241   LET
16514   38    A
16515   2Ø    =
16516   33    5
16517   126   "the next 5 bytes are a binary number"
16518   131┐
16519   32 │
1652Ø   Ø  │
16521   Ø  │  floating point binary version
16522   Ø  ┘  of the number 5
16523   118   NEWLINE
```

Or, you can look at the Display File by entering

```
PEEK 16396 + 256*PEEK 16397
```

as the start address ; the ZX81 can accept any valid expression as a response to an INPUT statement, this expression gives the current value of D-FILE; which is the address of the start of the Display File area of RAM. If the program has been entered exactly as shown, the result will be;

```
16581   118   NEWLINE - always present at start of display file
16582   29    1
16583   34    6
```

```
16584  33    5
16585  36    8
16586  29    1
16587  Ø     space
16588  29    1
16589  29    1
16590  36    8
16591  118   NEWLINE, end of first line of display
  --
```

Note that if you have more than 3 1/2 K of RAM, then the first line will be padded out with spaces (code Ø) to give a total of 32 characters before the NEWLINE code 118.

In this example output, the program is effectively displaying what it has already printed out ("16581 118" etc), but by adding a test line 5 to the program, you can see what that does to the Display File.

Variable Peeker

The program DECIMAL PEEKER can't be used as it stands to look at the variables area of RAM because the variables area keeps moving as the Display File is added to (at least, on a ZX81 with less than 3 1/2 K of RAM). To overcome this problem, the program has to re-calculate its reference point in the variables area for each value, resulting in the new program;

```
1Ø LET A=Ø
2Ø PRINT A;TAB 5;PEEK (A+PEEK 164ØØ+256*PEEK 164Ø1)
3Ø LET A=A+1
4Ø GOTO 2Ø
```

Adding a test line;

```
5 LET B=5
```

will give an output;

```
Ø    1Ø3    B (CODE "B" + 64)
1    131 ┐
2    32  │
3    Ø   │  Binary floating point '5'
4    Ø   │
5    Ø   ┘
6    1Ø2    A (CODE "A" + 64)
     --
```

Now try changing the test line 5 to set up a string variable, or an array, or a numeric variable with a multi-character name. Chapter 27 of the Sinclair ZX81 BASIC manual should help you to interpret the results.

RAMTEST 1K+

Ever had the feeling that your RAM was letting you down ? If so then this program may help. It does a quite thorough check of all the RAM fitted to your system except for that being used by the program, by POKEing random numbers into all of the locations to be tested and then PEEKing to see if they are all still correct. Having gone through the memory once in this fashion, it repeats the test with a different set of random numbers. There are some types of fault that - theoretically - this program won't detect, but they are very rare.

As listed the program will run for ever or until it finds an error - whichever is the sooner. If you omit line 14 then it will only make one 'pass', taking between 30 seconds (1K RAM) and 12 minutes (16K).

The listing below is for a ZX81 with 1K of RAM. Add 1024 to the second value (17351) in lines 7 & 11 for each additional 1K of RAM you have fitted. Also, the first value (16762) in lines 7 & 11 should be changed to 17530 if you have more than 3½K of RAM alltogether. Thus if you have 16K, lines 7 and 11 should read;

```
FOR C=VAL "17530" TO VAL "32711"
```

```
 1 FAST
 2 CLS
 3 LET A=VAL "256"
 4 RAND
 5 LET B=A*RND
 6 RAND B
 7 FOR C=VAL "16762" TO VAL "17351"
 8 POKE C,INT (A*RND)
 9 NEXT C
10 RAND B
11 FOR C=VAL "16762" TO VAL "17351"
12 IF PEEK C<>INT (A*RND) THEN STOP
13 NEXT C
14 RUN
```

ROMTEST 1K

This program - which takes about a minute to run - checks the contents of the ZX81's ROM for wrong bits.

The old 8K ROM (which insists that SQR .25 is 1.359) should give the answer 854885. The later ROM should give 855106.

```
10 FAST
20 LET A=0
30 FOR B=0 TO 8191
40 LET A=A+PEEK B
50 NEXT B
60 PRINT A
```

Renumber 1K

By J.Durst.

This program generates a machine code routine which will re-number the lines of a BASIC program, in multiples of 10.

As the length of the program has had to be kept to a minimum to allow it to be used on a 1K ZX81, it does not re-calculate line numbers after GOTO or GOSUB. It does, however, mark them by displaying these numbers in reverse video so they can be recognised easily.

To use the program;

- Enter ; POKE 16388,18Ø
  POKE 16389,67
  NEW
- LOAD the RENUMBER program, and RUN it.
- Enter NEW, to scrap the BASIC version of RENUMBER.
- LOAD the program to be renumbered.
- Enter; FAST
  PRINT USR 17332
  This should return 0 0/0 when the program has been re-numbered.
- Change the line numbers after any GOTOs or GOSUBs.

The changed program can now be saved, and another one loaded for re-numbering.

If you have more than 1K of RAM, then for each additional 1K; add 4 to the value 67 POKEd into 16389, add 1024 to the value 17332 in the direct PRINT USR command, and add 1024 to the value 17331 in line 40.

The machine code can be POKEd to give a different start line number or a different interval; POKE the desired values into locations 17333 (Start line) or 17362 (interval). These addr-esses should be increased by 1024 for each additional 1K of RAM.

```
   1Ø LET A$="11ØØØØ2AØC4ØØ17F4ØE
D42E5444D217D4ØE518Ø83E76EDB128Ø
2180DE521ØAØØ19EBE17223732B18EBE
1C13EECBE28Ø83CEDA128Ø5EØ18F323Ø
B3E25BE38EC7ECBFF77233E7EBE2ØF61
8EØ"
  2Ø FOR J=1 TO LEN A$/2
  3Ø LET X=J*2
  4Ø POKE 17331+J,CODE A$(X-1)*16+CODE A$(X)-476
  5Ø NEXT J
```

Catalogue 1K

A useful routine for those who - like the author - forget what names were given to programs saved on tape; this program reads the name from tape and displays it on the TV screen.

When you RUN the program the screen will go blank for a few seconds, then the L cursor will appear. Set up the cassette recorder as if to LOAD, then press NEWLINE. When the program name has been read it will be displayed - with the last character in inverse video - and the L cursor will re-appear ready to read another name.

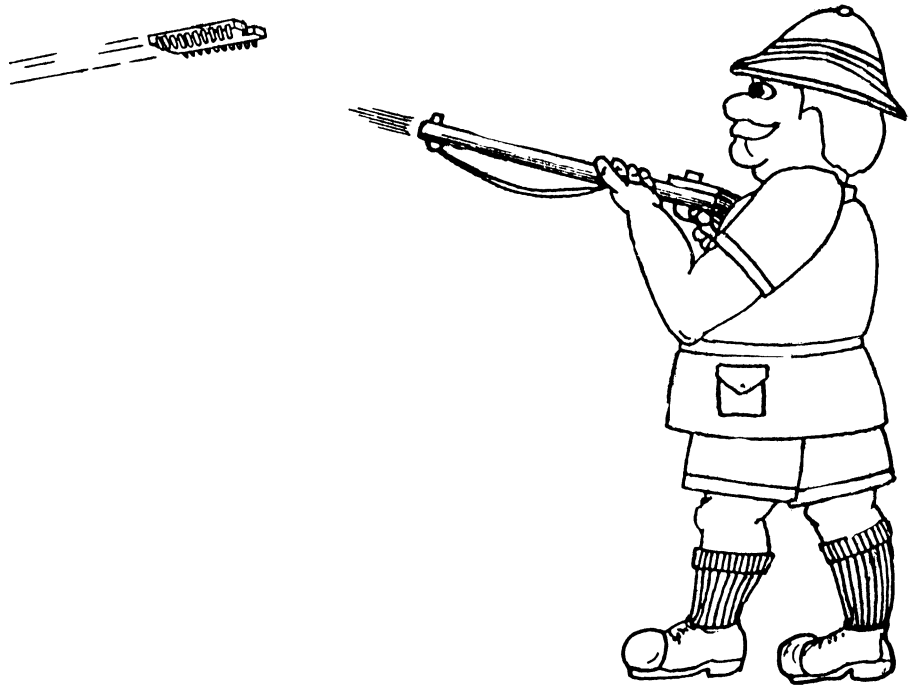
```
  1 REM 12345678901234567890123456789012345678900
    12345678901234567890123456
 10 LET A$="CD8740187FB0E0106003E7FDBFED3FF1FD017
    1738101F2F12A0C40CD87402371CB7928F7C9D51E94
    061A1DDBFE17CB7B7B38F510F5D12004FE5630CB3FCB
    1130C6C9"
 20 LET A=16514
 30 FOR B=1 TO 131 STEP 2
 40 POKE A,16*CODE A$(B)+CODE A$(B+1)-476
 50 LET A=A+1
 60 NEXT B
100 INPUT A$
110 DIM A$(32)
120 CLS
130 PRINT A$
140 RAND USR 16514
150 GOTO 100
```

The program is based on the machine code routine listed below, which is a modification of the normal ZX81's LOAD routine as described in the chapter "Discovering the ROM". Note that there are 66 characters after the REM in line 1, and 123 characters between quotation marks in line 10

```
4082  CD 87 40    CALL 4087    Wait for
4085  18 FB       JR 4082       silent period.
4087  0E 01       LD C 01
4089  06 00       LD B 00      Start to read 1 byte
408B  3E 7F       LD A 7F
408D  DB FE       IN A (FE)    Flash
408F  D3 FF       OUT (FF) A    TV
4091  1F          RRA          Check keyboard
4092  D0          RET NC       Quit if Break
4093  17          RLA          Go to
4094  17          RLA           40A7 if cassette
4095  38 10       JR C 40A7      input present.
4097  10 F2       DJNZ 408B    Back for next bit
```

```
4Ø99  F1          POP AF
4Ø9A  2A ØC 4Ø    LD HL (4ØØC)     Start of Disp File
4Ø9D  CD 87 4Ø    CALL 4Ø87        Get a byte.
4ØAØ  23          INC HL           Put it
4ØA1  71          LD (HL) C         onto screen.
4ØA2  CB 79       BIT 7,C          Get another
4ØA4  28 F7       JR Z 4Ø9D         if bit 7 not 1
4ØA6  C9          RET              Else return to BASIC.
4ØA7  D5          PUSH DE
4ØA8  1E 94       LD E 94          Start to get
4ØAA  Ø6 1A       LD B 1A           byte from tape.
4ØAC  1D          DEC E
4ØAD  DB FE       IN A (FE)        Decrement E reg
4ØAF  17          RLA               while tone burst
4ØBØ  CB 7B       BIT 7 E            being received.
4ØB2  7B          LD A E
4ØB3  38 F5       JR C 4ØAA        Jump if cass input
4ØB5  1Ø F5       DJNZ 4ØAC
4ØB7  D1          POP DE
4ØB8  2Ø Ø4       JR NZ 4ØBE
4ØBA  FE 56       CP 56
4ØBC  3Ø CB       JR NC 4Ø89       Save bit
4ØBE  3F          CCF               in C reg.
4ØBF  CB 11       RL C             Back for other
4ØC1  3Ø C6       JR NC 4Ø89        7 bits.
4ØC3  C9          RET              Return after 8th bit.
```

Chapter 7



Hardware

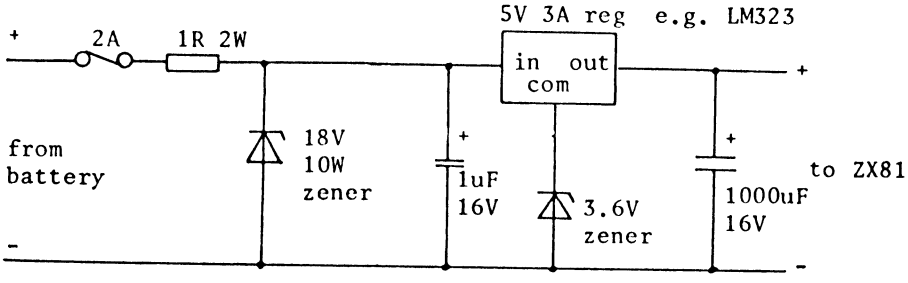
Hardware Hints

This section suggests some ways to improve the ZX81 hardware. Memory and I/O circuits are given at the end of this chapter.

Battery Power

Since portable televisions are available which will run off a 12 volt car battery, it would be nice to be able to run a ZX81 from the battery as well, for mobile computing perhaps, or for work during a power cut.

But, a car battery gives nominally 12 volts, which can rise to almost 15 volts when it is being charged heavily, and this is too high for the ZX81. To reduce the voltage to an acceptable level, use the circuit shown below, which will supply sufficient current (at least 1.2A) to drive a ZX81 fitted with a 16K RAM pack and the ZX printer.



The LM323 should be mounted on a heatsink; a piece of 1.5mm (1/16") aluminium 5cm x 8cm (2" x 3") is sufficient. The fuse, 1 ohm resistor and the 18 volt zener protect against voltage spikes coming from the battery lines, and also against reverse polarity connection, the 18V zener will not normally get hot so it does not need a heatsink.

This circuit is also useful if you suffer from frequent mains supply interruptions, which will 'zap' your program easily, as running your system from a car battery - which can either be recharged periodically or trickle charged all the time - should overcome the problem.

Cool it

All electronic equipment works more reliably if it is kept cool. Drilling a row of 1/4" holes along the rear face of the top moulding and some more in the base of the ZX81, near the front, will make a noticeable difference to the temperature inside the case. But take the case apart first - there are 5 fixing screws in the base, three being hidden by the feet - and carefully remove the printed circuit board first to avoid damaging it while drilling.

Getta Betta Jack

Some users have reported that the jack plugs provided on the cassette and power supply leads can cause problems because of poor connections. If this happens then it can be very annoying as even the shortest break in the power supply to the ZX81 will wipe out your program, while erratic cassette connections can ruin a recording or frustrate your attempts to LOAD. Replacing the plugs (they are 3.5mm types), or even soldering the wires directly to the ZX81 PCB, is the only solution.

Support your RAM

The main cause of problems with Sinclair's 16K RAM pack are the connections between it and the ZX81. Even the slightest momentary fault here is liable to corrupt your program or put the ZX81 into its notorious 'white-out' mode.

Unfortunately, the RAM pack is relatively heavy, and the ZX81 is mounted on resilient feet, so that any pressure on the ZX81's keyboard will rock the whole assembly, a procedure which is almost guaranteed to find any poor contacts ! The problem is made worse because the ZX81's contacts are only tinned, not gold plated, and so oxidise very easily.

Techniques used successfully by various ZX81 owners involve thickening the contacts by adding a layer of solder, adding a double-sided gold plated plug extension to the ZX81, cleaning the contacts regularly, fitting a supporting bracket between the ZX81 and the RAM pack to fix them rigidly together, and even soldering the two permanently together ! Whatever you choose to do, try not to remove the RAM pack too frequently as this only wears down the contacts on the ZX81 printed circuit board.

Spiky Mains

If your ZX81 keeps crashing or corrupting its program for no apparent reason, suspect large voltage spikes on the mains supply. They are usually caused by electric motors - as in your air conditioning, refrigerator or washing machine - being turned on or off. If you can identify which machine is causing the spikes then you may be able to plan your computing time along the lines of;

```
IF NOT WASHINGMACHINEON THEN COMPUTE
```

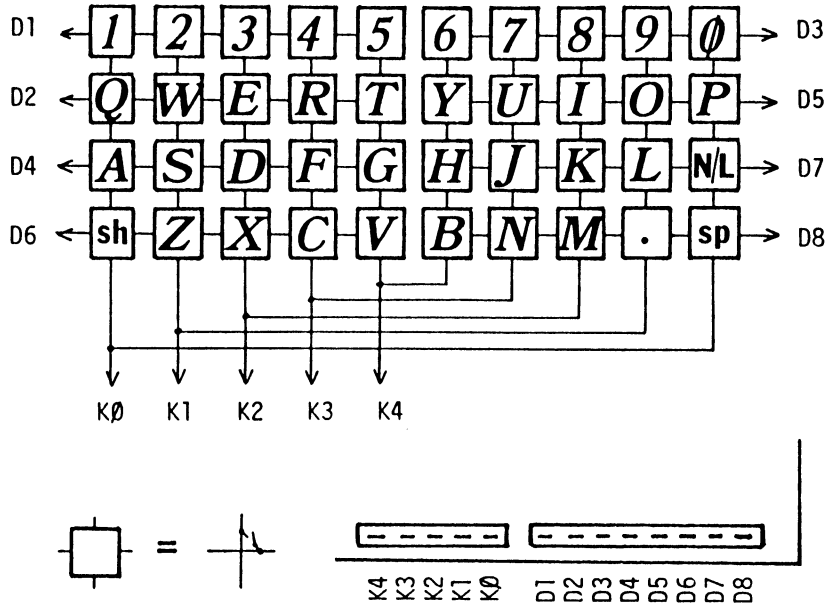
Using another mains outlet may help, or it may make things worse. The ultimate solution is to fit a mains filter in the supply lead to your ZX81 and TV, they are often advertised in most computing and electronics magazines.

Another Keyboard

The ZX81'a keyboard may be both cheap and aesthetically pleasing, but it is a pain (literally !) to use for any length of time.

Several firms now offer full-size add-on keyboards, but it is quite easy to make one for yourself from 40 keyswitches (single pole, normally open types), some wire, and some thought about how to mount the finished assembly.

There is no need to disconnect the ZX81's keyboard, the two can be used together, or if you can find a suitable small connector then the new keyboard can be made detachable. In any case, keep the wires as short as is reasonably possible, say 25cm (10") max.



Component side view of corner of ZX81 board showing keyboard connection sockets.

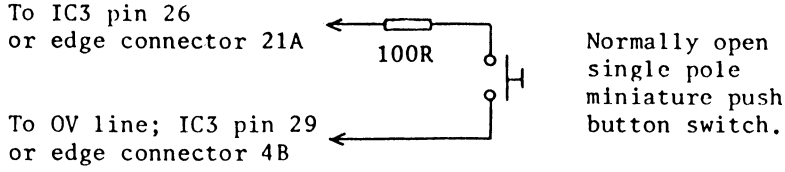
Push to Reset

If you are developing a Machine Language program, or attempting to load a poor quality tape, then you are liable to get your ZX81 into the notorious 'white-out' mode where the TV screen goes blank and none of the ZX81's keys has any effect.

The only way to recover on a standard ZX81 is by switching the power off momentarily. But this can be annoying, and - to the

author at least - it seems that repeatedly turning the mains on and off can't be good for the electronic circuits.

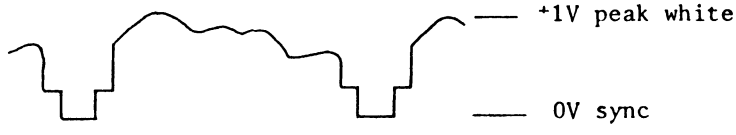
A more sanitary solution is to fit a small push-button switch to the rear of the ZX81's case, and wire it to the processor's RESET input as shown below. Pushing the switch will then put the ZX81 back into a responsive mode just as if you had interrupted the power supply, but in a gentler fashion.You will still lose the program from memory, but that would have happened anyway.



Connecting a Monitor

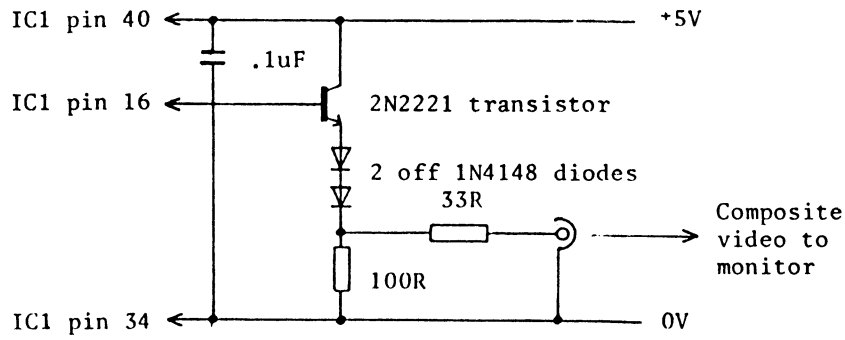
Although the ZX81 gives a good display on most TV sets once the tuning, contrast and brilliance controls have been adjusted, the picture definition is limited by the UHF modulation and demodulation processes, and particularly by the filters in the television receiver.

Anyone lucky enough to possess a monitor can get a much sharper picture by driving it with a composite (video plus sync) signal from the ZX81. Most monitors are designed to work with an input signal looking like;



and generally the impedance at the monitor's video input socket will be 75 ohms for feeding via co-axial cable.

The ZX81 can be modified to give a suitable composite video output by adding the circuit shown below. It picks the video signal from the SCL chip output, and buffers it so that the ZX81 UHF and TAPE outputs are not affected by the loading of the monitor input impedance. With care the components will fit between the ZX81's EAR socket and the UHF modulator box, and a phono socket can be mounted on the rear of the case.

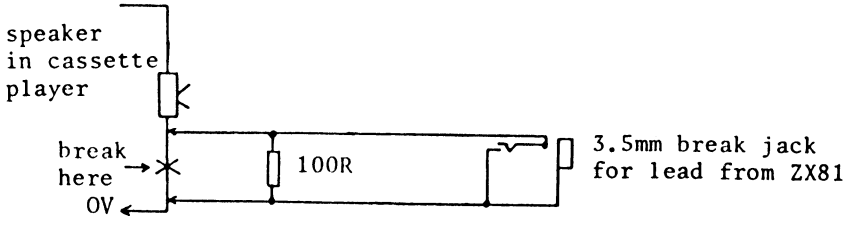


This circuit sometimes works better if you use only 1 diode.

From the Speaker

By connecting a suitable resistor and a jack socket in series with the loudspeaker of your cassette recorder, as shown below, you will be able to hear the signal while LOADing. This is a very useful facility when trying to LOAD from a difficult tape, as at least you can tell when the ZX81 *should* have finished LOADing.

Experiment with the value of the resistor to get the best level of sound from the speaker - or you could fit a 1K potentiometer connected as a variable resistor to allow you to vary the sound level as required.



Recording

There is nothing more aggravating that finding that the only tape you had made of an important program won't load. So it is well worth while investing a few hours in finding the most reliable settings for your cassette recorder controls and the

best brand of tape. (It is also worth getting into the habit of saveing more than one copy of each program !)

If you are still unable to SAVE and LOAD programs successfully despite experimenting with your recorder's level and tone controls, and even after trying several brands of tape, then;

- Try removing the EAR connector when SAVEing, and removing the MIC connector when LOADing. Some cassette recorders do not like simultaneous connections .
- If you have been running your recorder from mains, try if possible to run it from batteries to reduce any possible mains hum.
- If you are running your recorder from batteries then make sure that they are fresh. It is also worth noting that doing a fast rewind can temporarily deplete the batteries so in that case give them a minute or two to recover before starting to LOAD or SAVE.
- Make sure that the tape head is well past the end of the tape leader film - and allow the recorder a few seconds to pick up speed - before pressing NEWLINE when SAVEing.
- Make sure that the cassette recorder head is clean.
- The recorder should be at least one metre away from your television, monitor, or any other potential source of radiated interference. CB sets can sometimes be a nuisance, a Breaker's Break-in can get SAVEd along with your program and will totally confuse the ZX81 when you try to LOAD.
- Listen to the signal recorded onto the tape. The silent period should be *silent*, without any stray whistles, pops, crackles or frying noises. The high pitched buzz following should sound crisp and again should be free of any interfering noises. While you are doing this make sure that the plug connections are firm and that they and the recorder level and tone controls aren't responsible for adding any unwanted background to the signal.
- Some cheap cassette recorders use a permanent magnet to bias the record head and also for erasing the tape. With these, repeated recording on the same tape will build up a background hiss which the ZX81 can find disconcerting. Invest in a new tape, or have it wiped clean by a recorder with AC erase or on a bulk cassette eraser.

More esoteric causes of failure are poor azimuth (which means -roughly- that the record/playback head is in the wrong position relative to the tape) or DC magnetisation of the head, but these are unlikely. A few cassette players may not be able to give enough voltage to drive the ZX81 - which likes at least 0.5 volts r.m.s - in this case you could try using the speaker output as described previously.

Overall, the ZX81 likes a signal that is free of wow and flutter or any extraneous noise, from a cassette player that is running at somewhere near the correct speed - if you can LOAD your own recordings but have difficulty with other peoples', then suspect the record/playback head or the tape speed on your recorder.

Memory Map

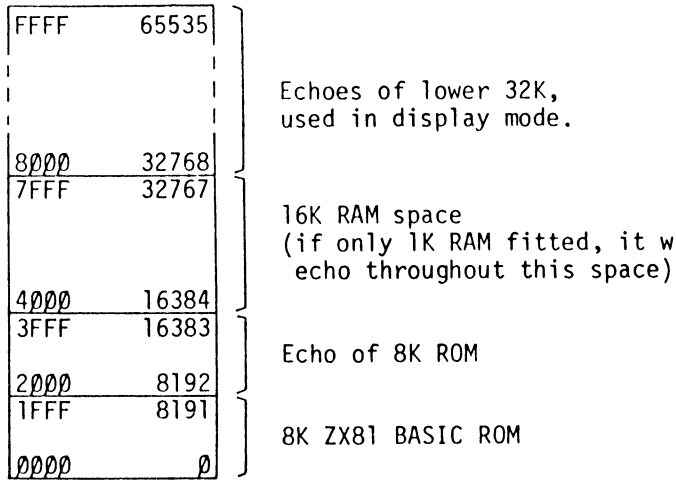
The ZX81's RAM and ROM address decoding uses the address lines AØ to A15 as follows;

ZX81 Address Line;	A15	A14	A13	A12	A11	A1Ø	A9	A8	-----	AØ
ROM		Ø		A	A	A	A	A	------	A
RAM 1K		1					A	A	-----	A
RAM 16K		1	A	A	A	A	A	A	------	A

Where the 'A's are the address lines used internally by the ROM or RAM chips.

A15 is brought to a '1' when the ZX81 is fetching a character to be displayed from RAM, at other times it is 'Ø'.

Because not all of the address lines are used, 'echoes' of the ROM and RAM appear throughout the 64K address space. The resulting memory map is;



There seems to be a lot of address space wasted on 'echoes', even when the 16K RAM pack is fitted, so it is interesting to speculate on how much memory could be fitted to a ZX81 without resorting to complicated 'bank-switching' schemes.

Starting with the basic 1K RAM ZX81, we can disable the internal RAM (to make room for bigger things) by simply connecting RAM.CS (the edge connector pin 2A) to +5V.

To remove the echoes of the 8K ROM, we have to disable it when either A15 or A13 are at '1'.

Having cleared the decks, let us see where we can put RAM, other than the 'traditional' 16K space from 4ØØØ to 7FFF. The 8K area 2ØØØ-3FFF seems a good choice, except that the ZX81 won't allow the BASIC program, variables or display file to be put here. It would be suitable, however, as a temporary storage area for data or machine code.

Above 8ØØØ however, we run into the ZX81's display generation mechanism. Basically this involves a machine code routine that;

- Puts the address of the start of the current line to be displayed into the Z80's HL register pair.
- Sets the most significant bit of the HL register pair to '1'
- Does a 'Jump' to the address in HL.

Special hardware then recognises that the Z80 is fetching a machine code instruction from an address with A15 set to '1', and does two things;

- It collects the byte read out from RAM (which will be from the display file) as being the next character to be sent to the TV display.
- It feeds a NOP (no operation) instruction to the Z80.

On seeing the NOP instruction, the Z80 increments the address to fetch the next instruction.

This goes on - the Z80 steadily fetching the characters to be displayed from RAM but seeing NOP codes instead - until a NEW LINE character is fetched.

What all this means in terms of adding memory is that the Z80 expects to see the display file area of RAM appear in two places; its 'normal' position and also some 32768 (8ØØØ hex) bytes higher when generating the display.

Luckily the echo of the display file is always accessed by a Z80 instruction fetch cycle, which is signalled by the M1 line from the Z80 going low. If we can be sure that the 'extra' RAM won't be accessed by a Z80 instruction fetch cycle, then we can use the M1 line to switch between the new RAM and the echo of the display file in lower RAM. We can be sure of this by keeping the new area of RAM free of machine code routines.

Summarising, we could put 56K of RAM onto a ZX81;

- 8K bytes from 2ØØØ to 3FFF, as a 'scratch pad' area for data or machine code. It won't be SAVEd.
- 16K bytes from 4ØØØ to 7FFF, with no restrictions on use.
- 32K bytes from 8ØØØ to FFFF, which can be used as an extension to the Variables area, but must not hold machine code. This area can be SAVEd in the normal way.

One word of caution however, the ZX81 would take about 20 minutes to SAVE or LOAD 48K bytes, and the chances of an error occuring during this length of time must be quite high !

Pseudo-ROM

The dot patterns for the displayable characters are held in a table in ROM, from 1E00 to 1FFF hex, eight bytes being used for each character. This gives a total of 64 characters and other symbols, the inverse (white on black) versions being produced by inverting the video signal within the SCL chip. The patterns are stored in order of their CODE, thus the character 'Ø', which has the code 28 (1C hex) is held in the eight ROM bytes from 1EE0 to 1EE7, since;

1EE0 = 1E00 + 8 x 1C

The hexadecimal and binary versions of these 8 bytes are shown alongside, the resulting pattern can be seen.

Addr.	Hex.	Binary
1EEØ	ØØ	ØØØØØØØØ
1EE1	3C	ØØ1111ØØ
1EE2	46	Ø1ØØØ11Ø
1EE3	4A	Ø1ØØ1Ø1Ø
1EE4	52	Ø1Ø1ØØ1Ø
1EE5	62	Ø11ØØØ1Ø
1EE6	3C	ØØ1111ØØ
1EE7	ØØ	ØØØØØØØØ

When the ZX81 is producing the display, it presents 13 bit addresses to the ROM to select the wanted 8-bit patterns. The low 9 address lines, which specify which character is to be displayed and which of the 8 scan lines is currently being generated, come from the SCL chip. The 4 high address lines come from the Z80 processor, and are actually bits 1 to 4 of the Z80's I register, which is set to 1E as part of a routine executed by the NEW command.

Because the starting address of the table is determined by the contents of the Z80's I register, we can alter it. As a demonstration, the following program displays 64 characters then loads the I register with Ø, making the ZX81 use the beginning of the ROM as a dot pattern table. Ten seconds later, the display will be returned to normal by restoring the correct value to the I register.

```
 1 REM 12345
 1Ø POKE 16514,62
 2Ø POKE 16515,Ø
 3Ø POKE 16516,237
 4Ø POKE 16517,71
 5Ø POKE 16518,2Ø1
1ØØ FOR A=Ø TO 63
11Ø PRINT CHR$ A;
12Ø NEXT A
13Ø RAND USR 16514
14Ø PAUSE 5ØØ
15Ø POKE 16515,3Ø
16Ø RAND USR 16514
```

The machine code routine used is;

```
3E ØØ     LDA Ø
ED 47     LD I A
C9        RET
```

As the beginning of the ZX81's ROM wasn't intended to be used as a dot pattern table, it gives a meaningless display ! But, if we were to add some RAM to the ZX81, in such a way that the

As the beginning of the 8K ROM wasn't intended to be used as a dot pattern table, you will get a meaningless pattern on the screen (although it can be a pretty one if you run the 'Starburst' program given elsewhere in this book at the same time). But, is we were to add some RAM to the ZX81, in such a way that the display generation circuits could access it as though it were ROM, then we could POKE any dot patterns we wanted into this new RAM, and so create our own character set. This could include shapes for games programs, mathematical or engineering symbols, or elements of high resolution drawings.

The circuit given on the next page shows how 1K of RAM can be added in this way. Because the address inputs to the ROM don't correspond exactly to the address lines present at the ZX81 rear connector (there are resistors in series with the lower 9 lines), it is best to fit the 'Pseudo-ROM' inside the ZX81's case, and directly wire all of the connections as shown. The eatra current taken from the +5V supply will make the regulator run slightly hotter, but you have drilled cooling holes in the case, haven't you ?

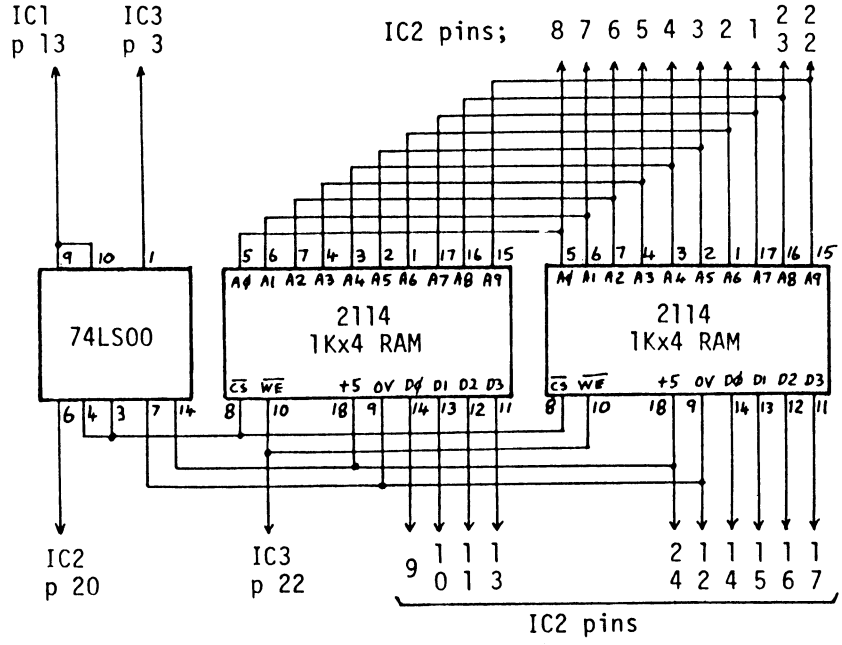
The Pseudo-ROM occupies addresses 2000-23FF hex (8192 to 9215 in decimal). This is enough for two sets of 64 character patterns, or you could load just one set of character patterns and have 512 bytes left for holding machine code or other data. Anything stored in these locations isn't SAVEd on tape, but neither is it affected by NEW or LOAD, so it is possible to run one program to fill the Pseudo-ROM, then LOAD another to use the new characters. You could also use this added RAM to hold data for transfer between programs.

To use the Pseudo-ROM to generate new characters, run the routine given on the previous page, but with the value POKEd by line 20 changed to 32 to select the lower half of the Pseudo-ROM, or 34 to select the upper half. Each character printed to the TV screen will then change to the new version.

But of course this will give a rubbish display unless you have POKEd meaningful dot patterns into the Pseudo-ROM. It is perhaps easiest to start by copying the normal dot patterns from the ZX81 ROM, by running a routine such as;

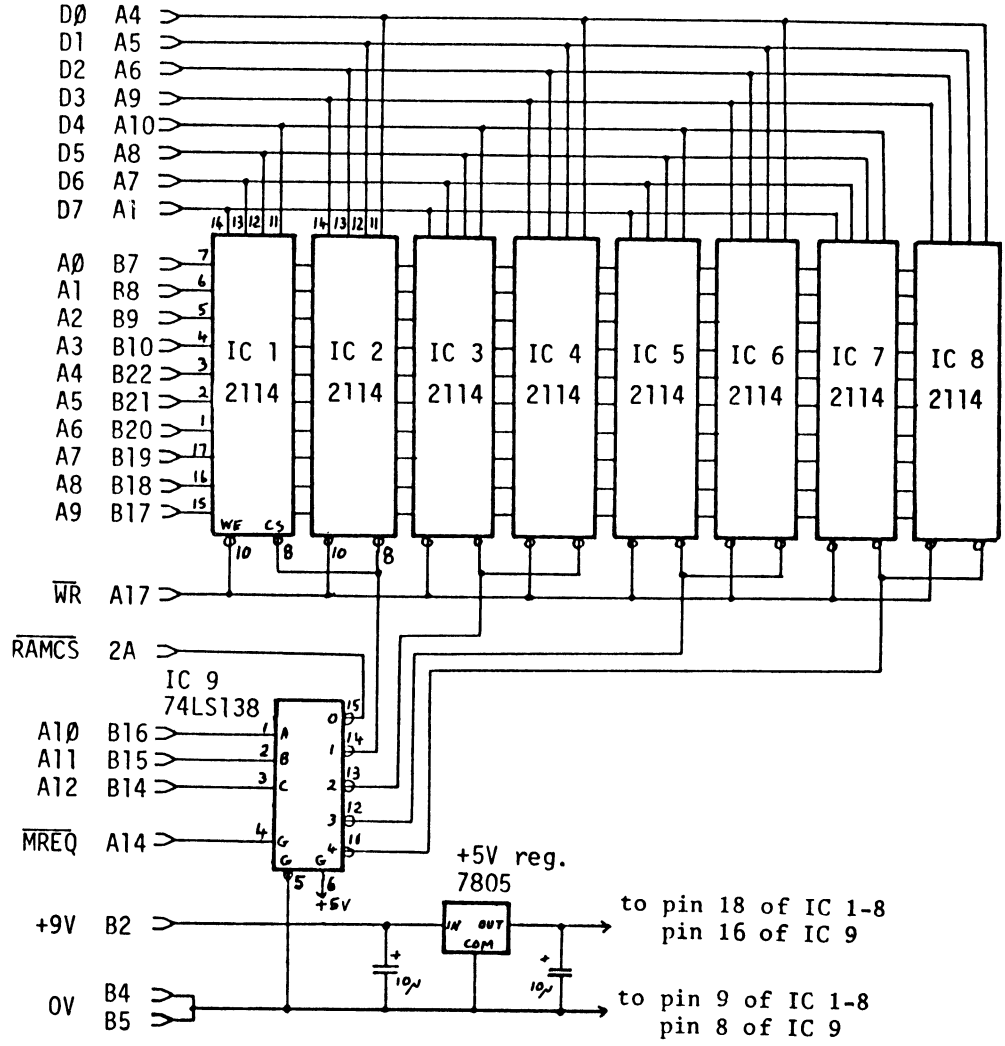
```
6Ø FOR A=Ø TO 511
7Ø POKE (8192+A),PEEK (768Ø+A)
8Ø NEXT A
```

You can then experiment with POKEing new patterns for particular characters; for example POKEing to locations 8416 to 8423 will affect the 'Ø' symbol on the display.



Note; R28 (680 ohm) to be removed from ZX81 board.

1-4K RAM Extension



This circuit is for a board which will plug onto the ZX81's rear connector, adding up to 4K of RAM to the existing 1K.

For 1K extension, fit ICs 1,2 and 9.
For 2K fit also ICs 3 and 4.
For 3K fit also ICs 3,4,5 and 6.
For 4K fit all ICs.

The 7805 5V regulator should be mounted on a 5cm. (2") square heatsink.

24 Line I/O

The circuit given on the following page is for an add-on board that will plug into the ZX81's rear connector and provide 24 bi-directional Input/Output lines. These could be used to control music synthesisers, model train layouts, or whatever you wish.

Because the ZX81 has 1K resistors in series with the data lines DØ-D7, which severely limit the possible DC and AC loading of these lines, a specialised MOS I/O chip has been used. This imposes virtually no DC load on the data lines and only a slight AC load. It is also an attractive choice because it is a very versatile IC, which can provide many types of I/O function.

As well as the 64K byte RAM/ROM address space, the Z80 processor can also access a separate 'I/O' address space. However, this I/O address space cannot be accessed directly from BASIC with PEEK or POKE statements, but needs a machine language routine. To simplify matters, the I/O port described here has been put into the Z80's normal RAM/ROM address space, at locations C000 to C003 hex, where it will not interfere with the ZX81's 8K ROM or up to 61K RAM. This address decoding is done by IC's 1,2 & 3.

If your ZX81 has only the basic 1K of RAM, then pin 8 of IC3 on the I/O board should be connected to pin A2 of the ZX81 rear connector; the RAMCS line. If, however, you have an external RAM pack, then you should;

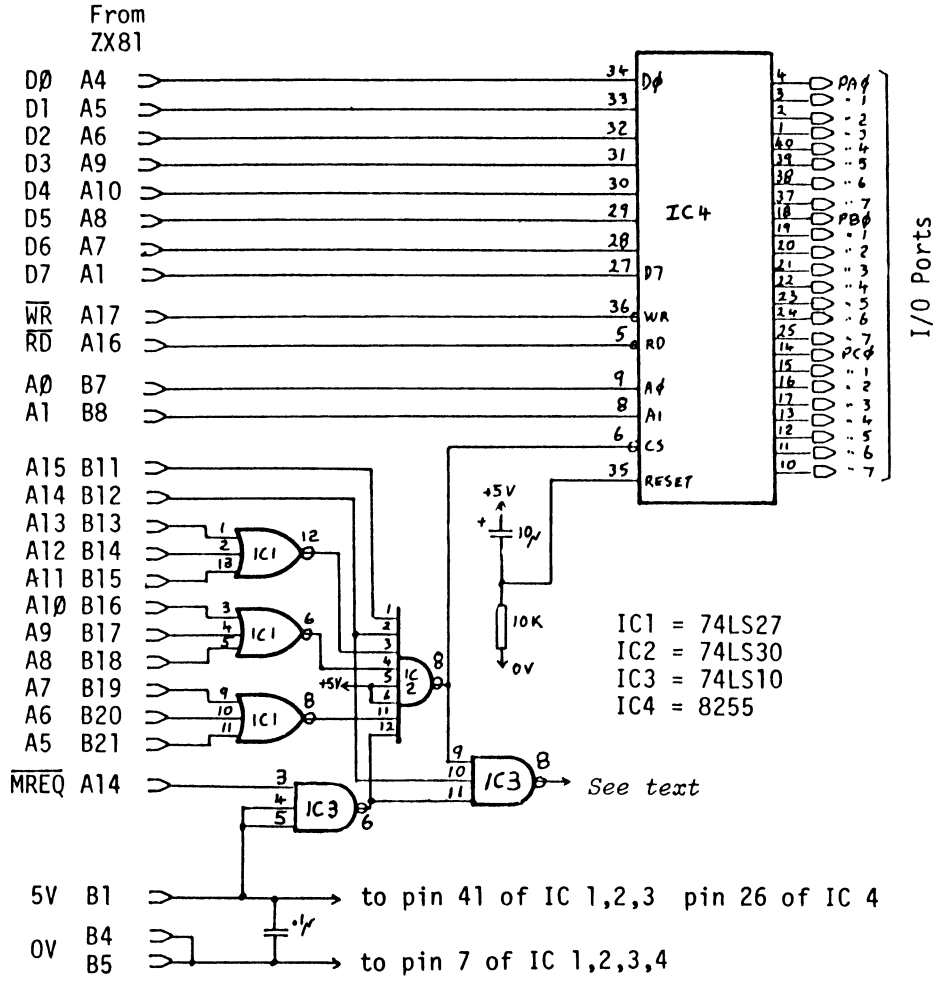
- Add a 23+23 way double sided plug to the I/O board so that the RAM pack can be plugged on. All pins *except for pin A14* on this plug should be wired to the corresponding pins of the I/O board's 23+23 way socket.
- Connect pin 8 of IC3 on the I/O board to pin A14 of the plug which will mate with the RAM pack.
- Connect pin 10 of IC3 to pin 9 of IC3 instead of to pin 2 of IC3.

If you have a printer then this can be connected between the ZX81 and the I/O board.

The following paragraphs summarise the main features of the 8255 Programmable Peripheral Interface chip, giving enough information for most purposes. For more complex applications, you would need to refer to the 8255 data sheets.

The 8255 has three 8-bit I/O ports (A,B and C), which can be written to (when they are in the output mode) or read from as sigle byte registers. It also has an internal 8-bit control register which can only be written to. In the circuit shown, the addresses of the four registers are;

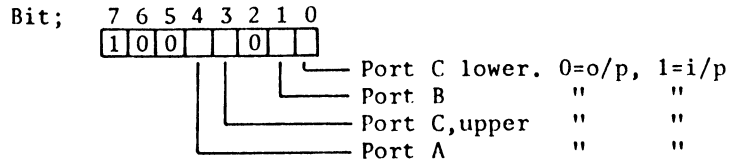
	Hex	Decimal
Port A	C000	49152
Port B	C001	49153
Port C	C002	49154
Control	C003	49155



It may be set up - by writing special codes to the control register - to one of three modes. Mode 0 is the simplest, and suitable for most applications.

In mode 0, ports A and B may each be set up as 8 input lines or as 8 output lines. Port C is divided into upper and lower 4-bit halves, each of which may be set up to act as input or output.

To set up the 8255, we need to write into the control register a byte constructed as;



Thus to set Port A for output, B for input, and both parts of Port C for input, we need to write an 8-bit pattern 10001011 to the Control register.

Remembering that;

If bit 7 = 1	this	corresponds	to	decimal	128
" " 6 = 1	"	"	"	"	64
" " 5 = 1	"	"	"	"	32
" " 4 = 1	"	"	"	"	16
" " 3 = 1	"	"	"	"	8
" " 2 = 1	"	"	"	"	4
" " 1 = 1	"	"	"	"	2
" " 0 = 1	"	"	"	"	1

then we can set up the 8255 by the BASIC statement;

```
POKE 49155, 139      (139 = 128+8+2+1)
```

Having set up the 8255, we can now send a pattern to the output port A by;

```
POKE 49152,X         (X is the pattern; 0-255)
```

The pattern on the 8 input lines connected to the B port can be read by;

```
LET A=PEEK 49153
```

and similarly, all 8 port C inputs can be read by;

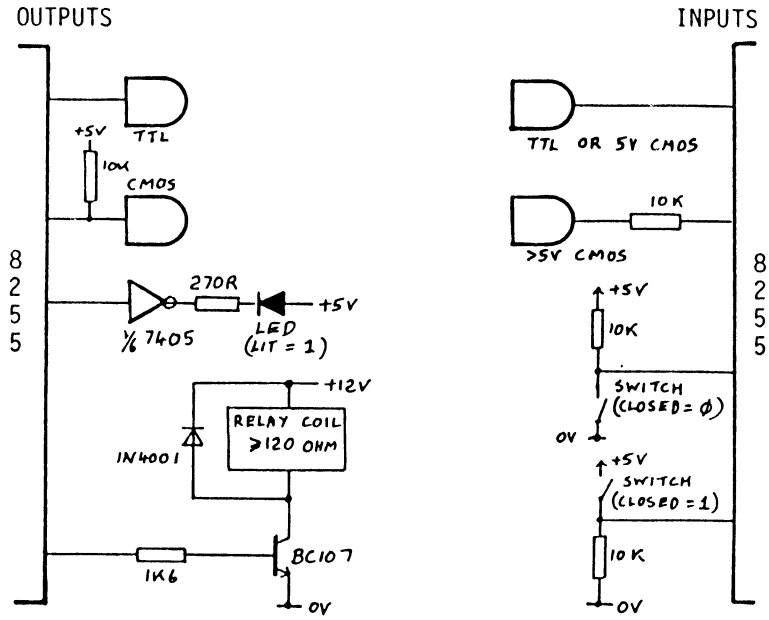
```
LET A=PEEK 49154
```

If either half of the C port has been set up for output, then any of the lines may be set or reset individually at any time by POKEing one of the following values to the Control register;

Value POKEd	Effect
0	PC0 reset to 0
1	PC0 set to 1
2	PC1 reset to 0
3	PC1 set to 1
4	PC2 reset to 0
5	PC2 set to 1
6	PC3 reset to 0
7	PC3 set to 1

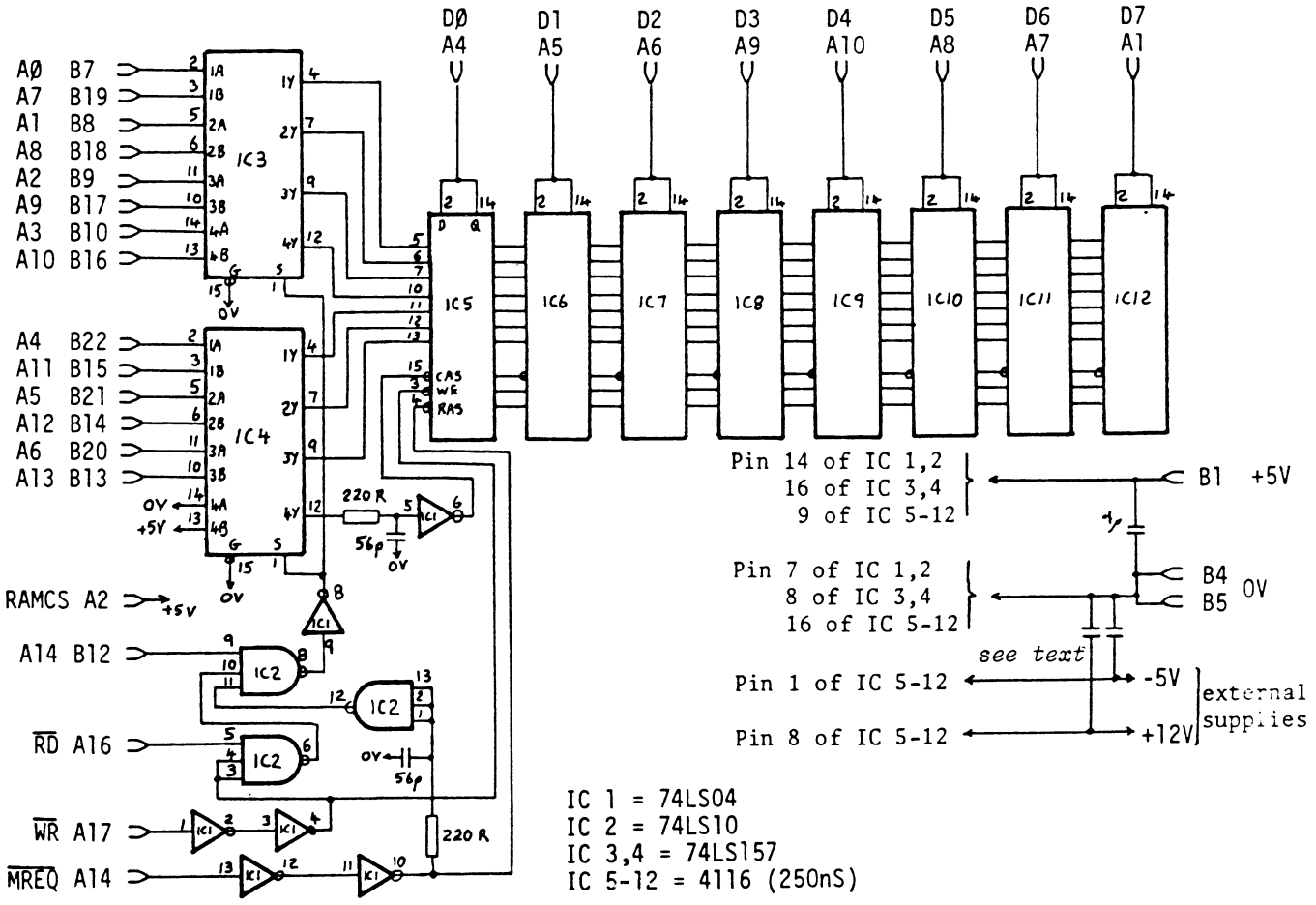
8	PC4 reset to 0
9	PC4 set to 1
10	PC5 reset to 0
11	PC5 set to 1
12	PC6 reset to 0
13	PC6 set to 1
14	PC7 reset to 0
15	PC7 set to 0

When set up as output lines, the 8255 outputs can each drive one standard TTL load. When acting as inputs, they present a high impedance to the driving source, and accept an input voltage of less than 0.8V as a '0', or a voltage between 2.0 and 5.0V as a '1'. Some useful interface circuits are shown below;



16K Byte RAM

The circuit on page 154 is for a 16K byte RAM extension board to plug onto the ZX81's rear connector. Connecting the board disables the ZX81's internal 1K RAM, but it is compatible with the Pseudo ROM and 24 line I/O circuits given earlier.


DØ A4
D1 A5
D2 A6
D3 A9
D4 A10
D5 A8
D6 A7
D7 A1
IC5
IC6
IC7
IC8
IC9
IC10
IC11
IC12
IC3
IC4
AØ B7
A7 B19
A1 B8
A8 B18
A2 B9
A9 B17
A3 B10
A10 B16
A4 B22
A11 B15
A5 B21
A12 B14
A6 B20
A13 B13
220 R
56p
RAMCS A2
A14 B12
RD A16
WR A17
MREQ A14
Pin 14 of IC 1,2
16 of IC 3,4
9 of IC 5-12
B1 +5V
Pin 7 of IC 1,2
8 of IC 3,4
16 of IC 5-12
B4 0V
B5
see text
Pin 1 of IC 5-12
-5V
Pin 8 of IC 5-12
+12V
external supplies
IC 1 = 74LS04
IC 2 = 74LS10
IC 3,4 = 74LS157
IC 5-12 = 4116 (250nS)

4116 dynamic RAM ICs have been used for low cost, but they do require additional - externally generated - supplies of +12V at 150mA and -5V at 5mA.

Because of the relatively poor noise immunity of the ZX81's data bus lines, and also because of the large current spikes drawn by the RAM chips, the constructor should make the board as small as is reasonably possible, with short connections. The 0V, -5V and 12V supplies should be fed in a 'grid' fashion to the 4116 chips, and a 0.1uF capacitor connected between the 0V and 12V lines, and another between the 0V and -5V lines, at alternate 4116's. Also, a 10uF solid tantalum bead capacitor should be connected between 0V and 12V, and another between 0V and -5V.

If all is well, then when the extension board is connected to the ZX81 and power applied the familiar K cursor should appear on the screen and all previous programs should run without difficulty The RAMTEST program given elsewhere in this book can then be run to check the new memory thoroughly.

BookWare™ *For the Timex Sinclair 1500, Timex Sinclair 1000 and ZX-81 Computers*

Here's your chance to explore the Timex Sinclair 1500 and discover new features and techniques. You also learn machine code programming and elements of RAM. The **Explorer's Guide** gives you 30 games, actual programs, and useful applications, routines and programming utilities. Special hardware tips include using battery power, connecting to a monitor, and adding a fullstroke keyboard.

The series includes:

Software

Monarch!
Demolition/Ten-Pin
Championship Chess
CasinoPak I: One-Arm Bandit/ Blackjack
The Nowotnik Puzzle
Invasion Force
Escape from Shazzar!
Math Master: Self-Teaching Software for the Timex Sinclair 1500/1000

Books

The Timex Sinclair 1500/1000 Pocket Book
BASIC Basics of the Timex Sinclair 1500/1000
50 1K/2K Games for the Timex Sinclair 1500/1000
Basics of Timex Sinclair 1500/1000 BASIC
49 Explosive Games for the Timex Sinclair 1500/1000
Mastering Machine Code on Your Timex Sinclair 1500/1000
Making the Most of Your Timex Sinclair 1500/1000
Explorer's Guide to the Timex Sinclair 1500/1000
Timex Sinclair 1500/1000 Machine Language Programming and Interfacing
Exploring Timex Sinclair 1500/1000 Graphics

Reston Publishing Company, Inc.
A Prentice-Hall Company
Reston, Virginia

0-8359-1816-5

0 21898 18165 7